1993

Structure editors and attribute grammar programming.

Farook A. Wadia
University of Windsor

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Structure Editors and Attribute Grammar Programming

by

Wadia Farook A.

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the Degree of
Master of Science at the
University of Windsor
Windsor, Ontario, Canada
1993

ISBN 0-315-87381-7

Canada

Name: WADIA FAROOK A

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

COMPUTER SCIENCE — SUBJECT TERM

0984 — SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

- Architecture 0729
- Art History 0377
- Cinema 0900
- Dance 0378
- Fine Arts 0357
- Information Science 0723
- Journalism 0391
- Library Science 0399
- Mass Communications 0708
- Music 0413
- Speech Communication 0459
- Theater 0465

EDUCATION

- General 0515
- Administration 0514
- Adult and Continuing 0516
- Agricultural 0517
- Art 0273
- Bilingual and Multicultural 0282
- Business 0688
- Community College 0275
- Curriculum and Instruction 0727
- Early Childhood 0518
- Elementary 0524
- Finance 0277
- Guidance and Counseling 0519
- Health 0680
- Higher 0745
- History of 0520
- Home Economics 0278
- Industrial 0521
- Language and Literature 0279
- Mathematics 0280
- Music 0522
- Philosophy of 0998
- Physical 0523
- Psychology 0525
- Reading 0535
- Religious 0527
- Sciences 0714
- Secondary 0533
- Social Sciences 0534
- Sociology of 0340
- Special 0529
- Teacher Training 0530
- Technology 0710
- Tests and Measurements 0288
- Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

- Language
  - General 0679
  - Ancient 0289
  - Linguistics 0290
  - Modern 0291
- Literature
  - General 0401
  - Classical 0294
  - Comparative 0295
  - Medieval 0297
  - Modern 0298
  - African 0316
  - American 0591
  - Asian 0305
  - Canadian (English) 0352
  - Canadian (French) 0355
  - English 0593
  - Germanic 0311
  - Latin American 0312
  - Middle Eastern 0315
  - Romance 0313
  - Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

- Philosophy 0422
- Religion
  - General 0318
  - Biblical Studies 0321
  - Clergy 0319
  - History of 0320
  - Philosophy of 0322
- Theology 0469

SOCIAL SCIENCES

- American Studies 0323
- Anthropology
  - Archaeology 0324
  - Cultural 0326
  - Physical 0327
- Business Administration
  - General 0310
  - Accounting 0272
  - Banking 0770
  - Management 0454
  - Marketing 0338
- Canadian Studies 0385
- Economics
  - General 0501
  - Agricultural 0503
  - Commerce-Business 0505
  - Finance 0508
  - History 0509
  - Labor 0510
  - Theory 0511
- Folklore 0358
- Geography 0366
- Gerontology 0351
- History
  - General 0578
  - Ancient 0579
  - Medieval 0581
  - Modern 0582
  - Black 0328
  - African 0331
  - Asia, Australia and Oceania 0332
  - Canadian 0334
  - European 0335
  - Latin American 0336
  - Middle Eastern 0333
  - United States 0337
- History of Science 0585
- Law 0398
- Political Science
  - General 0615
  - International Law and Relations 0616
  - Public Administration 0617
- Recreation 0814
- Social Work 0452
- Sociology
  - General 0626
  - Criminology and Penology 0627
  - Demography 0938
  - Ethnic and Racial Studies 0631
  - Individual and Family Studies 0628
  - Industrial and Labor Relations 0629
  - Public and Social Welfare 0630
  - Social Structure and Development 0700
  - Theory and Methods 0344
- Transportation 0709
- Urban and Regional Planning 0999
- Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

- Agriculture
  - General 0473
  - Agronomy 0285
  - Animal Culture and Nutrition 0475
  - Animal Pathology 0476
  - Food Science and Technology 0359
  - Forestry and Wildlife 0478
  - Plant Culture 0479
  - Plant Pathology 0480
  - Plant Physiology 0817
  - Range Management 0777
  - Wood Technology 0746
- Biology
  - General 0306
  - Anatomy 0287
  - Biostatistics 0308
  - Botany 0309
  - Cell 0379
  - Ecology 0329
  - Entomology 0353
  - Genetics 0369
  - Limnology 0793
  - Microbiology 0410
  - Molecular 0307
  - Neuroscience 0317
  - Oceanography 0416
  - Physiology 0433
  - Radiation 0821
  - Veterinary Science 0778
  - Zoology 0472
- Biophysics
  - General 0786
  - Medical 0760

EARTH SCIENCES

- Biogeochemistry 0425
- Geochemistry 0996
- Geodesy 0370
- Geology 0372
- Geophysics 0373
- Hydrology 0388
- Mineralogy 0411
- Paleobotany 0345
- Paleoecology 0426
- Paleontology 0418
- Paleozoology 0985
- Palynology 0427
- Physical Geography 0368
- Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

- Environmental Sciences 0768
- Health Sciences
  - General 0566
  - Audiology 0300
  - Chemotherapy 0992
  - Dentistry 0567
  - Education 0350
  - Hospital Management 0769
  - Human Development 0758
  - Immunology 0982
  - Medicine and Surgery 0564
  - Mental Health 0347
  - Nursing 0569
  - Nutrition 0570
  - Obstetrics and Gynecology 0380
  - Occupational Health and Therapy 0354
  - Ophthalmology 0381
  - Pathology 0571
  - Pharmacology 0419
  - Pharmacy 0572
  - Physical Therapy 0382
  - Public Health 0573
  - Radiology 0574
  - Recreation 0575
  - Speech Pathology 0460
  - Toxicology 0383
- Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences

- Chemistry
  - General 0485
  - Agricultural 0749
  - Analytical 0486
  - Biochemistry 0487
  - Inorganic 0488
  - Nuclear 0738
  - Organic 0490
  - Pharmaceutical 0491
  - Physical 0494
  - Polymer 0495
  - Radiation 0754
- Mathematics 0405
- Physics
  - General 0605
  - Acoustics 0986
  - Astronomy and Astrophysics 0606
  - Atmospheric Science 0608
  - Atomic 0748
  - Electronics and Electricity 0607
  - Elementary Particles and High Energy 0798
  - Fluid and Plasma 0759
  - Molecular 0609
  - Nuclear 0610
  - Optics 0752
  - Radiation 0756
  - Solid State 0611
- Statistics 0463

Applied Sciences

- Applied Mechanics 0346
- Computer Science 0984
- Engineering
  - General 0537
  - Aerospace 0538
  - Agricultural 0539
  - Automotive 0540
  - Biomedical 0541
  - Chemical 0542
  - Civil 0543
  - Electronics and Electrical 0544
  - Heat and Thermodynamics 0348
  - Hydraulic 0545
  - Industrial 0546
  - Marine 0547
  - Materials Science 0794
  - Mechanical 0548
  - Metallurgy 0743
  - Mining 0551
  - Nuclear 0552
  - Packaging 0549
  - Petroleum 0765
  - Sanitary and Municipal 0554
  - System Science 0790
- Geotechnology 0428
- Operations Research 0796
- Plastics Technology 0795
- Textile Technology 0994

PSYCHOLOGY

- General 0621
- Behavioral 0384
- Clinical 0622
- Developmental 0620
- Experimental 0623
- Industrial 0624
- Personality 0625
- Physiological 0989
- Psychobiology 0349
- Psychometrics 0632
- Social 0451

ABSTRACT

Attribute grammars provide a formal and yet intuitive way of specifying the static semantics of programming languages, but their use is not limited to compiler generation systems. Attribute grammars have been successfully used to provide solutions to problems from areas such as natural language processing, SQL processors, and circuit design transformers within a VLSI package [25]. One of the major advantage of using attribute grammar as a programming paradigm is the modular and declarative structure that results. Also, programs written as executable specification of attribute grammars are easy to debug because they mirror the structure of the input data.

W/AGE is an attribute grammar programming environment. It supports the attribute grammar programming paradigm where programs are constructed as executable specifications of attribute grammars. It consists of several functions that extend the standard environment of the pure lazy functional programming language Miranda.

Programs written in W/AGE have to obey many syntax and layout rules; also, in general, as the size of the program and the number of attributes used increases it may be difficult to keep track of the synthesized and inherited attributes used in the program. Compiling a W/AGE program containing syntax errors and missing attribute definitions can lead to errors which are at times very difficult to debug

and at times can be very much frustating.

The purpose of this thesis is to built a structure editor for W/AGE and evaluate its usefulness in developing attribute grammar specifications.

This work is dedicated to my parents, aunt Suraiya and Waheeda, without their inspiration and support I would not have been what I am today.

Acknowledgments

I owe the successful completion of my thesis work to many people who in some way or the other have helped me reach my goal.

First of all, I am indebted to my parents who have always prayed for my continued success and well-being.

I cannot miss this opportunity to express my sincere gratitude and thanks to Dr. Richard Frost, my supervisor. Working with him was a wonderful experience, his support and guidance made my stay in Canada very comfortable.

Dr. Subir Bandyopadhyay, my co-supervisor, I guess if it would not have been for him I would have left back for India the very next day I landed in Canada. His critical comments and advice regarding my thesis work and otherwise, have proven to be very useful and helped me make important decisions.

I extend my gratitude to Dr Gold, my external examiner, for being on my committee and providing valuable comments.

And last but not the least, I would like to thank all my peers in the graduate lab, Steve Karamatos, Walid Nyamneh, our wonderful secretaries Mary and Margaret and Dr. Morrissey for all your help. Thanks for being there.

TABLE OF CONTENTS

Chapter 1 INTRODUCTION

§ 1.1 Motivation for the Thesis

Attribute grammars provide a formal and yet intuitive way of specifying the static semantics of programming languages and as such have been used extensively in applications such as compiler generators, language-based editor generating tools and so on. Attribute grammars have been successfully used to provide solutions to problems from areas such as natural language processing, database query processors, and circuit design transformers [24]. One of the major advantages of using attribute grammars as a programming paradigm is the modular and declarative structure that results. Also, programs written as executable specification of attribute grammars are easy to reason about because they mirror the structure of the input data.

Many systems supporting the attribute grammar programming paradigm have been built [25]. In most of these systems, non-executable specifications of attribute grammars are compiled into code in some conventional host programming language. The pre-processing and translation of code into the host programming language results in increased indirection in the programming environment which may be tolerated less in problem domains where the use, and the advantages, of the attribute grammar formalism are not so obvious [23]. Extending the environment of conventional programming languages to support attribute grammar

constructs is a non-trivial task. Also, since most of the conventional programming languages use *strict* (non-lazy) evaluation strategy, a good deal of unnecessary computation of attributes is required. Also, in order to support fully general attribute dependencies either substantial transformation or multi-pass evaluation is required.

W/AGE [25] is an attribute grammar programming environment. It supports the attribute grammar programming paradigm allowing programs to be constructed as executable specifications of attribute grammars. It consists of several functions that extend the standard environment of the pure, lazy functional programming language Miranda™. The problems which are encountered in supporting the attribute grammar paradigm in a conventional programming language do not arise in lazy functional programming language. Attribute grammar constructs can be easily added by using higher-order functions [23] and also because of lazy evaluation no unnecessary computation of an attribute value is required until it is deemed necessary. In addition, neither transformation nor multiple evaluations are required to support general attribute dependencies.

The resulting combination of attribute grammar and functional programming paradigm facilitates software development in several ways:

• Reasoning about programs for the purpose of verification, complexity analysis, transformations, *etc.*, becomes easy, because the programs are extremely

declarative, modular and variable free.

• The combined paradigm lends itself well to the technique of deriving *program from proofs*.

• Programs are easy to debug, modify and maintain, because the structure of the program closely resembles the structure of the input data.

But the advantages derived from the combined paradigm are sometimes offset by the fact that programs written in W/AGE have to obey complex syntax and layout rules. Also, in general, as the size of the program and the number of attributes used increases, it may be difficult to keep track of the synthesized and inherited attribute definitions in the program. Compiling a W/AGE program containing syntax errors and missing attribute definitions can lead to errors which can be extremely difficult to debug and could intimidate a novice programmer.

A software tool, such as a *structure editor*, for W/AGE can alleviate the problem of syntax errors and missing attribute definitions by checking for them as the program is being developed. As a result, after compiling the program, the programmer can spend less time debugging syntax errors and missing attribute definitions, and can concentrate more on the design and development of the program.

§ 1.2 Thesis Statement

The purpose of the thesis work is to support my argument in the previous

section that a structure editor for W/AGE would be a useful software tool to develop executable specification of attribute grammars and to show that structure editors are relatively easy to build and to use.

Therefore, I propose the following thesis:

It is possible, within the time constraints of Master's thesis work, to build a structure editor for W/AGE which:

• detects all syntax errors, errors in the use of attributes, facilitates in the development of executable specification of attribute grammars, and

• can be mastered by programmers, with only one or two years of programming experience in less than a week.

§ 1.3 Why is the thesis important ?

There is a growing interest amongst software developers in interactive program development tools which would facilitate the development and management of software. Structure editors for different phases of a software development cycle have been built and are a part of some commercially available software development environments. However, there is very little specific information available that indicates the difficulty of building and using structure editors. The establishment of the thesis will provide such information and will be of value to organizations who are considering the use of structure editors in their operations.

§ 1.4 Work Done

In order to support and test the thesis, the following work was carried out:

• A survey of structure editors was carried out. The survey covered amongst other things, structure editors and structure editor generators that were built for research and pedagogical purpose, and together with ones that are commercially available.

• A pure, lazy functional programming language Miranda; the attribute grammar programming paradigm; and W/AGE — an attribute grammar programming environment (built by extending the standard environment of Miranda™) were studied.

• A structure editor generating tool, the **Cornell Synthesizer Generator**, was studied.

• A structure editor, **WAGE-ed**, for W/AGE was built using the **Cornell Synthesizer Generator**.

• Users of **WAGE-ed** were surveyed and the results analysed.

Chapter 2 STRUCTURE EDITORS

§ 2.1 What is a Structure Editor ?

The need to manage the development of large software systems is one of the most pressing problems faced by computer programmers. An important aspect of this problem is the design of new tools to aid interactive program development.

Recently, research on structure editing has developed promising ways to enhance the power of tools used by programmers.

Structure editors are specialized editors which have the knowledge of the syntax and semantics of the underlying language. Meyrowitz and van Dam [15] define structure editing as manipulation of general structures such as trees and graphs instead of raw text. They further define syntax-directed editing or language-based editing as a subset of structure editing where the general structures represent particular syntactic structures of the language being edited.

A language-based editor makes use of the context-free syntax of the programming language to ensure that the program being developed is syntactically correct and well-formed at all times. It reinforces the view that a program is a hierarchical composition of computational structures. Programs are composed of *templates*, which provide predefined, formatted patterns for each of the language constructs. Programs are created top-down by inserting new templates at *placeholders* in the

skeleton of previously entered templates. A program being developed using a language-based editor is represented by its derivation tree. On the other hand, a text-oriented editor has no knowledge about the syntax and semantics of the underlying language they edit.

Language-based editors naturally support interaction. Data development and incremental computation can be performed at edit time. Text editors support only data development, whereas language-based editors support both data development and program execution and can therefore be considered as complete interactive programs rather than just an editors.

Also, a language-based editor alleviates some of the conceptual misunderstandings that occur between the user and the system. Using a text editor the programmer may write some program text with one meaning in mind, whereas the system may interpret it differently. The dangling ***else*** problem is a good example. If language-based editors are used to develop programs, fewer misunderstandings occur.

§ 2.2 Benefits of Structure Editing

Apart from its main purpose of facilitating interactive program development, there are other benefits which derive from structural editing. A structure editor,

- Ensures that the program is structurally well-formed at all times,

• Prohibits the user from making inappropriate insertions by restricting the choices offered to the insertions that are legal in the context of the current selection,

• Guarantees the syntactic integrity of the program at every step by enforcing the removal and insertion of entire, well-formed, program fragments,

• Provides a mechanism for making controlled changes in a single step through *transformation* operations,

• Allows a program to be displayed according to its hierarchical structure,

• Performs automatic indentation of programs,

• Prohibits typographical errors in structural units because the templates are predefined and immutable,

• Allows programs to be developed at a higher-level of abstraction, because templates that correspond to abstract computational units are inserted and removed as units,

• Can also allow text editing to be integrated, but at the same time preclude the creation of syntactically incorrect programs.

§ 2.3 Survey on Structure Editors

Work on structure editors was inspired by research work carried out in the 1970's on program analysis, transformations and translation facilities for a language-based editor for a programming language of the Algol family and also on

LISP environments which have long exploited the ability to manipulate programs as data objects.

The Mentor [16] project, which was initiated in 1974, created a collection of special-purpose tools for processing PASCAL abstract syntax trees using a general purpose tree-manipulation language. Around 1979, three systems, *viz.*, Gandalf, Lispedit and the Cornell Program Synthesizer delved into ways of unifying language-based program editing with execution and debugging.

To date, a great deal of research work has been done in this field and a large collection of published work is available. The result of a survey work carried out in this field is described in [56], it lists and describes various (research and commercial) structure editors that have been built, their features, structure editor generating tools (research and commercial), structure editor generating technologies, *etc.*

§ 2.4 The Cornell Synthesizer Generator

The **Cornell Synthesizer Generator** [48] is a tool for creating language-based editors. Just as a parser generator may be used to create a parser from a grammar that specifies the language's concrete syntax, the Cornell Synthesizer Generator can be used to create a language-based editor given a specification of the language's abstract syntax, context-sensitive relationship, display format, concrete input syntax, and transformation rules for restructuring programs. From

these specifications, the Synthesizer Generator creates a language-based editor for manipulating objects according to these rules.

The editor specifications are written in SSL (Synthesizer Specification Language), which is itself built around the concept of attribute grammars and a type definition facility.

The Synthesizer Generator is written in C and runs under Berkeley UNIX. It can generate editors for the X Window System, SunView and video display terminals.

Chapter 3 ATTRIBUTE GRAMMAR PROGRAMMING

§ 3.1 What is an attribute grammar ?

Definition

An *attribute grammar* is a generalization of a context-free grammar.

Attribute grammars were first proposed by Knuth in 1968 as a notation for specifying the static semantics of programming languages. An attribute grammar is a syntax-directed definition where each grammar symbol[1] (nonterminals) has an associated set of attributes. These attributes are partitioned into two disjoint sets: *synthesized* attributes and *inherited* attributes.

The value of an attribute associated with a grammar symbol at a parse-tree node is defined by semantic rules associated with the production used at that node. The value of a synthesized attribute at a node in the parse-tree is a function of the value of the attributes at that node and/or at the children of that node. The value of an inherited attribute at a node in the parse-tree is a function of the value of the attributes at the parent and/or the sibling nodes.

---

1 In an attribute grammar definition, terminals are assumed to have synthesized attributes only and the root nonterminal cannot have inherited attributes.

§ 3.2 Attribute Grammar Programming

In an attribute grammar definition, each production in the grammar of the form $A \longrightarrow \beta$ has associated with it a set of semantic rules. Each semantic rule is of the form $a = f(a_1, a_2, ..., a_k)$ where f is a function, and either

- a is a synthesized attribute of A and $a_1, a_2, ..., a_k$ are attributes associated with grammar symbols appearing on the right side of the production and /or A, or
- a is an inherited attribute of one or more grammar symbols appearing on the right side of the production and $a_1, a_2, ..., a_k$ are attributes associated with A or grammar symbols appearing on the right side of the production.

Example

Consider writing an attribute grammar specification which accepts two parameters, one a number (say n) and second a list of numbers (say *ls*). The problem is to identify how many occurrences of the number n are present in the list *ls*. An attribute grammar specification involving synthesized and inherited attributes is shown on Figure 3.1.

It should be noted that this is not the only attribute grammar solution to this problem but has been chosen for explanatory purposes.

```
input ::= number ":" "[" list_of_numbers "]"
          NUM_VAL v list_of_numbers = VAL ^ number
          NUM_OCCUR ^ lhs = NUM_OCCUR ^ list_of_numbers

list_of_numbers ::= number "," list_of_numbers
                    NUM_VAL v list_of_numbers = NUM_VAL v lhs
                    NUM_OCCUR ^ lhs = add_num_occur[VAL ^ number,
                                      NUM_OCCUR ^ list_of_numbers,
                                      NUM_VAL v lhs]
                    | number
                      NUM_OCCUR ^ lhs =
                                init_num_occur[VAL ^ number,
                                               NUM_VAL v lhs]
```

Figure 3.1

In Figure 3.1 above:

• The text in boldface is a context-free grammar which specifies the structure of the input string. The notation used is a variant of BNF, where terminal symbols appear in quotes,

• The symbol ^ signifies that the attribute is synthesized. For example, *NUM_OCCURS* ^ ***list_of_numbers*** should be read as 'the *NUM_OCCURS* attribute that is passed up/synthesized for the nonterminal ***list_of_numbers***',

• The symbol v signifies that the attribute is inherited. For example, *NUM_VAL* v ***list_of_numbers*** should be read as 'the *NUM_VAL* attribute passed down/inherited to/by the nonterminal ***list_of_numbers***',

• The text which appear in italics are semantic rules. The semantic rules indicate how the value of a synthesized or inherited attribute is obtained. For example, in the semantic rule "*NUM_OCCUR* ^ ***lhs*** = init_num_occur[*VAL* ^ ***number***, *NUM_VAL* v ***lhs***]", the attribute *NUM_OCCUR* passed up ***lhs*** (list_of_numbers)

is obtained by applying the function **init_num_occur** to the *VAL* attribute passed up ***number*** and the attribute *NUM_VAL* passed down to ***lhs***.

The advantage of using attribute grammars as a programming paradigm is the *modular* structure that results [23]. Modular design is one of the most desired feature of a software system. It allows parts of a system to be easily extended and/or reused. For example, the attribute grammar specification in the Figure 3.1 could be easily extended to process a list of strings. The extended specification is shown in Figure 3.2.

```
input ::= number ":" "[" list_of_numbers "]"
          NUM_VAL v list_of_numbers = VAL ^ number
          NUM_OCCUR ^ lhs = NUM_OCCUR ^ list_of_numbers
        | string ":" "[" list_of_strings "]"
          STR_VAL v list_of_strings = VAL ^ string
          NUM_OCCUR ^ lhs = NUM_OCCUR ^ list_of_strings

list_of_numbers ::= number "," list_of_numbers
                    NUM_VAL v list_of_numbers = NUM_VAL v lhs
                    NUM_OCCUR ^ lhs = add_num_occur[
                                      VAL ^ number,
                                      NUM_OCCUR ^ list_of_numbers,
                                      NUM_VAL v lhs]
        | number
          NUM_OCCUR ^ lhs = init_num_occur[VAL ^ number,
                                           NUM_VAL v lhs]

list_of_strings ::= string "," list_of_strings
                    STR_VAL v list_of_strings = STR_VAL v lhs
                    NUM_OCCUR ^ lhs = add_str_num_occur[
                                      VAL ^ string,
                                      NUM_OCCUR ^ list_of_strings,
                                      STR_VAL v lhs]
        | string
          NUM_OCCUR ^ lhs = init_str_num_occur[VAL ^ string,
                                               STR_VAL v lhs]
```

Figure 3.2

As shown by the above example, it is very easy to extend existing specifications. Another advantage inherent in this programming paradigm is that there is a clean separation between syntax and semantics, and the computation of attributes is well-structured. The programmer could first specify the overall structure of a passage without having to worry about the associated semantic actions. This leads to modular design of programs and which are easy to debug and test because they reflect the structure of the input data.

Chapter 4 W/AGE — Windsor Attribute Grammar Environment

W/AGE supports the attribute grammar programming paradigm by extending the environment of a pure, lazy functional programming language Miranda[2] [25]. Attribute grammar specifications written in W/AGE are declarative, modular and variable free.

§ 4.1 Structure of a W/AGE Program

An attribute grammar specification written in W/AGE consists of various sections:

File inclusion section

This section specifies the files which are to be included when the W/AGE program is compiled. If the full pathname is included between angle brackets then the file is searched in the Miranda directory. If the pathname is specified between double quotes then the file should reside in the current working directory. The following file must be included, using **%insert**, in all W/AGE programs:

%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>

---

2 Miranda is a trademark of Research Software Ltd.

Attribute definition section

This section is used to declare the name and type of attributes that are used in the program.

Special symbol declaration section

All single character symbols, other than alphabets and digits, that are used in the program must be declared within single quotes in this section.

Reserved word declaration section

All words that are to be treated as reserved words in the program, *e.g.* language keywords, must be defined within double quotes in this section.

Interpreter definition section

This section contains definition of interpreters (nonterminals) that are used in the program. An interpreter could be defined as one of the following types: ***literal***, ***interpreted***, ***uninterpreted***, ***recognised***, or ***structure***.

Each production of an interpreter of type ***structure*** may have optional semantic rules. In W/AGE, there are three types of semantic rules, as explained later, in section 4.3.6.2.

If any Miranda functions are used in a W/AGE program[3], then they may be defined in the same file which contains the W/AGE program or they may be

[3] Note that the Miranda functions are not a part of W/AGE.

defined in a separate file in which case the name of the file must be specified in the program using the **%insert** command. All files that are specified in the W/AGE program using the **%insert** command are included during compilation.

§ 4.2 Components of W/AGE

W/AGE consists of the following components:

- A function for applying interpreters: ***apply_interpreter***,
- A lexical scanning function: ***tokenise***,
- A set of functions for building basic interpreters: ***literal***, ***interpreted***, and ***uninterpreted***,
- A set of interpreter combinators: ***$orelse***, ***$excl_orelse***, ***$enables***, and ***structure***.

For more information on W/AGE refer to Appendix 3.

§ 4.3 Support for Left-Recursive Interpreter Definitions in W/AGE

According to a widely held belief, it is not possible to construct executable specifications of language processors that use a top-down parsing strategy and which have structures that directly reflect the structure of grammars containing left-recursive productions. It has been shown in [24] that top-down parsing and left-recursive productions can co-exist. The technique involves the use of non-

left-recursive recognizers, known as *guards*, to avoid non-termination of top-down left-recursive language processors. The description below is summarised from [24].

W/AGE provides a function ***enables*** to support left-recursive interpreter definitions. Each left-recursive interpreter definition, **f** = **e**, where **f** is of the form **f —> f *** is replaced by a guarded-left-recursive definition **f = r $enables e'**, where **r** is a non-left-recursive recognizer and **r** and **e** recognize the same language. And **e'** is obtained from **e** by replacing each left-recursive alternative ρ in **e** by **p $enables** ρ, where **p** is a non-left-recursive recognizer, recognizing the same language as ρ.

The current version of W/AGE supports left-recursive interpreter definitions. For more information, the interested reader is referred to [24].

§ 4.4 An example of attribute grammar specification in W/AGE

The W/AGE program for the attribute grammar specification listed in Figure 3.2 is given below:

```
-----------------------------------------------------
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
|| ATTRIBUTE DECLARATION
```

Figure 4.1 (Continued . . .)

```
attribute
::=
   LITERAL_VAL terminal
   | NUM_VAL num
   | STR_VAL [char]
   | NUM_OCCUR num
|| SPECIAL SYMBOLS DECLARATION
special_symbols
=
 [',',':','[',']']
|| RESERVED WORD DECLARATION
reserved_words
=
 []
|| INTERPRETER DEFINITIONS
any_number
=
 literal INT_TERM
any_string
=
 literal IDENTIFIER_TERM
colon
=
 uninterpreted (SPECIAL_SYMBOL_TERM ":")
comma
=
 uninterpreted (SPECIAL_SYMBOL_TERM ",")
open_br
=
 uninterpreted (SPECIAL_SYMBOL_TERM "[")
close_br
=
 uninterpreted (SPECIAL_SYMBOL_TERM "]")
input
=
 structure (s1 any_number ++ s2 colon ++ s3 open_br
            ++ s4 list_of_numbers ++ s5 close_br)
  [a_rule 1.1 (NUM_VAL $d s4) EQ
              conv_2_NUM_VAL[LITERAL_VAL $u s1],
   c_rule 1.2 (NUM_OCCUR $u lhs) EQ (NUM_OCCUR $u s4)]
  $excl_orelse
  structure (s1 any_string ++ s2 colon ++ s3 open_br
            ++ s4 list_of_strings ++ s5 close_br)
  [a_rule 1.3 (STR_VAL $d s4) EQ
              conv_2_STR_VAL[LITERAL_VAL $u s1],
   c_rule 1.4 (NUM_OCCUR $u lhs) EQ (NUM_OCCUR $u s4)]
list_of_numbers
=
 structure (s1 any_number ++ s2 comma ++
            s3 list_of_numbers)
  [c_rule 2.1 (NUM_VAL $d s3) EQ (NUM_VAL $d lhs),
   a_rule 2.2 (NUM_OCCUR $u lhs) EQ
              add_NUM_OCCUR[LITERAL_VAL $u s1,
                            NUM_OCCUR $u s3,
                            NUM_VAL $d lhs]]
  $excl_orelse
```

Figure 4.1 (Continued . . .)

```
structure (s1 any_number)
[a_rule 2.3 (NUM_OCCUR $u lhs) EQ
            init_NUM_OCCUR[LITERAL_VAL $u s1,
                           NUM_VAL $d lhs]]
list_of_strings
=
structure (s1 any_string ++ s2 comma ++
           s3 list_of_strings)
[c_rule 2.1 (STR_VAL $d s3) EQ (STR_VAL $d lhs),
 a_rule 2.2 (NUM_OCCUR $u lhs) EQ
            add_STR_NUM_OCCUR[LITERAL_VAL $u s1,
                              NUM_OCCUR $u s3,
                              STR_VAL $d lhs]]
$excl_orelse
structure (s1 any_string)
[a_rule 2.3 (NUM_OCCUR $u lhs) EQ
            init_STR_NUM_OCCUR[LITERAL_VAL $u s1,
                               STR_VAL $d lhs]]
|| MIRANDA FUNCTIONS
add_STR_NUM_OCCUR[LITERAL_VAL (IDENTIFIER_TERM a),
                  NUM_OCCUR b,
                  STR_VAL c]
= NUM_OCCUR (1 + b) , a = c
= NUM_OCCUR b       , otherwise

add_NUM_OCCUR[LITERAL_VAL (INT_TERM a),NUM_OCCUR b,
              NUM_VAL c]
= NUM_OCCUR (1 + b)   , (numval a) = c
= NUM_OCCUR b         , otherwise

init_STR_NUM_OCCUR[LITERAL_VAL (IDENTIFIER_TERM a),
                   STR_VAL b]
= NUM_OCCUR 1    , a = b
= NUM_OCCUR 0    , otherwise

init_NUM_OCCUR[LITERAL_VAL (INT_TERM a),NUM_VAL b]
= NUM_OCCUR 1 , (numval a) = b
= NUM_OCCUR 0 , otherwise
conv_2_NUM_VAL[LITERAL_VAL (INT_TERM a)]
= NUM_VAL (numval a)

conv_2_STR_VAL[LITERAL_VAL (IDENTIFIER_TERM a)]
= STR_VAL a

------------------------------------------------------------
```

Figure 4.1

Chapter 5 WAGE-ed — A STRUCTURE EDITOR FOR W/AGE

§ 5.1 Introduction to WAGE-ed

WAGE-ed is an interactive program development tool which facilitates in the development of executable specification of attribute grammars. It was constructed in entirety by the author as part of the thesis work. A program being developed using **WAGE-ed** is continuously checked for syntax errors and missing attribute definitions. If any syntax errors and/or missing attribute definitions are detected they are immediately displayed on the terminal screen to provide feedback to the programmer as the program is developed and modified.

The knowledge, about W/AGE, incorporated in **WAGE-ed** prohibits the user from developing syntactically incorrect specifications, notifies the user of missing attribute definitions, performs transformation, pretty-printing and analysis of the object being edited. The editor checks the objects being edited for inconsistencies, prompts the user of the editor with legal alternative and/or imposes constraints on how the user can proceed.

WAGE-ed reinforces the view that a program is a hierarchical composition of computational structures. The editor provides templates which are predefined, formatted patterns for each construct in W/AGE, *e.g.*, it provides templates for entering interpreter definitions, attribute declarations and so on. The attribute

grammar specification is created top-down by inserting new templates at place-holders in the skeleton of previously entered templates. For example, in the figure shown below,

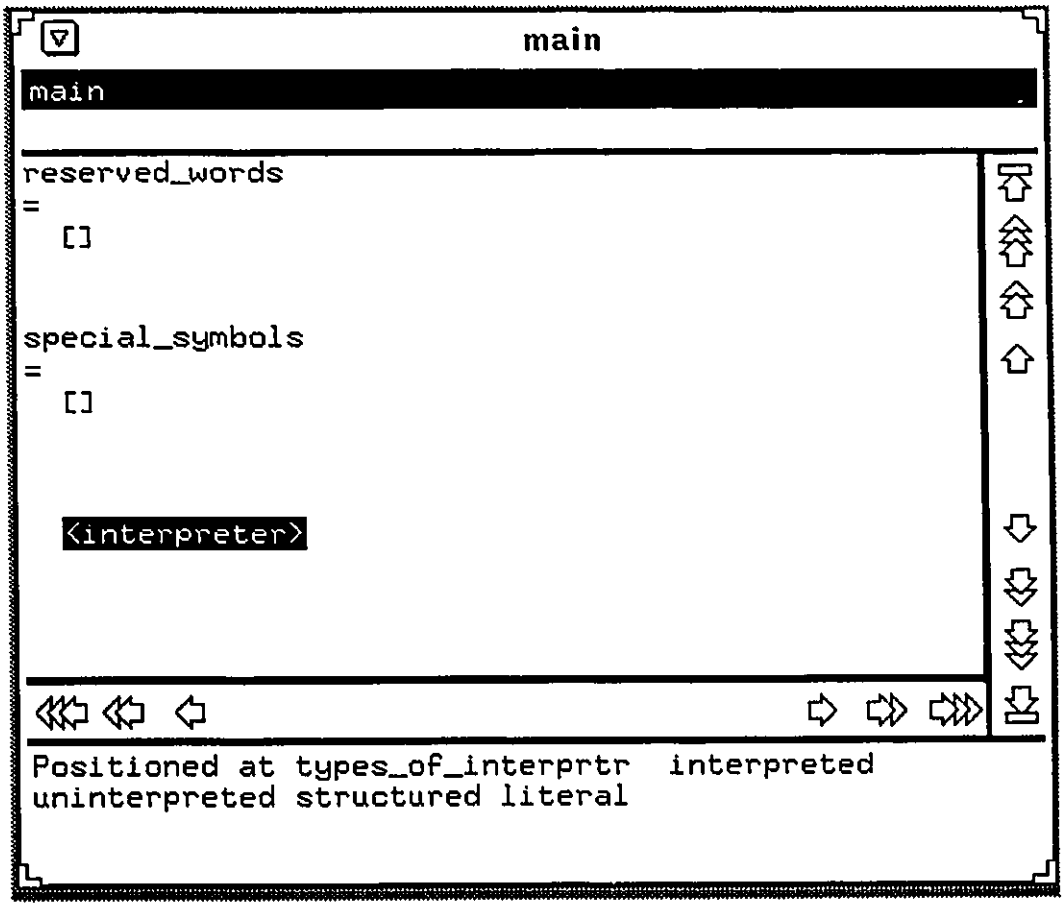

Figure 5.1

<interpreter> is a placeholder that identifies a location where additional insertions can be made.

When a program is being edited, it is read into a distinct *buffer*. Each window shows the contents of a distinct buffer; the same buffer can be displayed in more than one window. The window may be scrolled up or down to view different regions of the buffer.

The program contained in a buffer is a *term*, *i.e.*, derivation tree with respect to the underlying abstract syntax of W/AGE. The nodes of a term are instances of *operators* and the subtrees of a node are the operator's arguments, which are themselves terms. Each term has a two-dimensional, textual display representation. The view of a term displayed in a window is a rectangular section of this textual representation.

The program being edited in a buffer has a current *selection* (*i.e.*, insertion point). The selection can only be moved from one template to another, or from one template to its constituents, not from character to character nor from one line of text to another. The selections in **WAGE-ed** are indicated on the display screen by highlighting the selected region. There are two ways for making an object the current selection *viz.*,

- by selecting different tree-walking commands from the menu such as ***forward-sibling-with-optionals***, ***backward-sibling-with-optionals***, *etc.*, which allow navigation according to the structure of the term. These commands are also bound to a sequence of return keys,

• by using a mouse or cursor keys of an ASCII terminal. Clicking the mouse on a character causes the selection to change to the subterm associated with that character.

The net effect of each editing operation is the replacement of a selected subterm with another. The editing operations are carried out by using transformations or text editing.

Additional templates can be inserted into the program by choosing an item from a list of legal items displayed in the help pane. For example, as shown in Figure 5.1, if the current selection is <*interpreter*>, it can be transformed into a template for ***interpreted*** interpreter by selecting interpreted from the list of choices displayed. After making the selection, the <*interpreter*> template shown in Figure 5.1 above is transformed into one shown below:

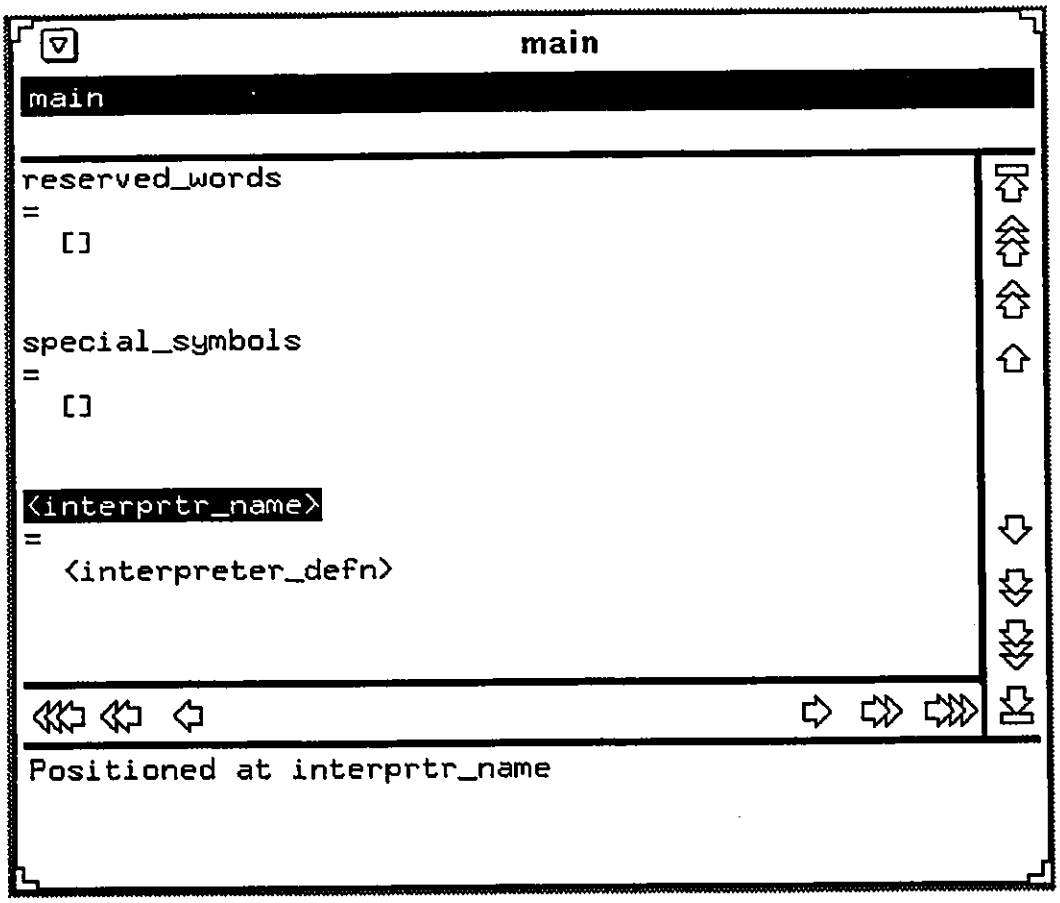

Figure 5.2

Notice that the placeholder *<interpreter_defn>* has been automatically indented according to the layout rules of Miranda.

WAGE-ed forbids the user from making inappropriate choices by offering only those choices which are legal in the context of current selection.

WAGE-ed allows changes to a program to be made by insertion and removal of entire, well-formed, program fragments. This ensures the syntactical integrity

of the program. Construct-to-construct transformation mechanism provided by **WAGE-ed** allows the user to construct the program at a higher-level of abstraction.

§ 5.2 Features of WAGE-ed

WAGE-ed eliminates much of the mundane task of program development and lets the programmer focus on the intellectually challenging aspect of programming. Its features include the following:

• Minimal text insertion: the user will need text only to enter interpreter names, attribute names, reserved words, special symbols, function names, and file names. **WAGE-ed** performs checking of lexical syntax, inserts all punctuation symbols, W/AGE keywords, *etc.*

• Automatic indentation: **WAGE-ed** indents the program according to the layout rules of Miranda. Because of automatic indentations certain type error messages are eliminated during compilation.

• Checking for missing attribute definitions: **WAGE-ed** continuously checks the program for attributes used but not declared in attribute declaration section, using synthesized/inherited attributes, in function calls, that have not been passed up/down the associated interpreter, multiple attribute declarations, *etc.*

Compiling an attribute grammar specification, which has been developed using **WAGE-ed** and has no warning messages, will compile without signalling any syntax errors or missing attribute definitions.

§ 5.3 Versions of WAGE-ed

There are two versions of **WAGE-ed**, namely:

• Guarded editors: a family of editors which support left-recursive interpreter definitions.

• Unguarded editors: a family of editors which do not support left-recursive interpreter definitions.

In each version, there are five editors which are classified according to their functionality. The following are the five editors, in each version, with their functionality:

• **ed_syntax_only**: attribute grammar specifications developed using this editor are guaranteed to be syntactically correct and conforms to the layout rules of Miranda. This editor does not check for any other kinds of errors.

• **ed_no_synh_inh**: attribute grammar specifications developed using this editor are guaranteed to be syntactically correct. In addition, this editor also checks for various other errors *e.g.* multiple attribute declarations in attribute declaration section.

This editor does not check for the use of inherited and/or synthesized attribute values which are not evaluated or missing attribute equations for synthesized and/or inherited attributes in semantic rules.

• **ed_synh**: this editor is similar in functionality to **ed_no_synh_inh**, except that in addition it keeps a check on the use of synthesized attribute values which are not evaluated or missing synthesized attribute definitions in semantic rules.

• **ed_inh**: this editor is similar in functionality to **ed_no_synh_inh**, except that in addition it keeps a check on the use of inherited attribute values which are not evaluated or missing inherited attribute definitions in semantic rules.

• **wage_editor**: this is a fully-fledged editor for W/AGE. Attribute grammar specifications developed using this editor are guaranteed to be syntactically correct, are checked for a number of different types of errors including the use of missing inherited and/or synthesized attribute values or missing inherited and/or synthesized attribute equations for synthesized and/or inherited attributes in semantic rules.

§ 5.4 Limitations of WAGE-ed

No Type Checking

Attribute grammar specifications developed using **WAGE-ed** are not checked for type errors. For example, if the following specification, which contains a type error, is entered using **WAGE-ed** the type error will not be reported by **WAGE-ed**.

```
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
attribute
::=
   LITERAL_VAL terminal
   | HOLD_SUM num
   | TOTAL_SUM num
special_symbols
=
 [',']
reserved_words
=
 []
comma
=
 uninterpreted (SPECIAL_SYMBOL_TERM ",")
a_number
=
 literal INT_TERM
list_of_numbers
=
 structure (s1 a_number ++ s2 comma ++ s3 list_of_numbers)
 [a_rule 1.1 (HOLD_SUM $d s3) EQ add_numbers[LITERAL_VAL $u s1,
                                             HOLD_SUM $d lhs],
  c_rule 1.2 (TOTAL_SUM $u lhs) EQ (TOTAL_SUM $u s3)]
 $excl_orelse
 structure (s1 a_number)
 [a_rule 1.3 (TOTAL_SUM $u lhs) EQ add_last_num[LITERAL_VAL $u s1
                                                HOLD_SUM $d lhs]
input
=
 structure (s1 list_of_numbers)
 [i_rule 2.1 (HOLD_SUM $d s1) EQ (HOLD_SUM 'o'), || ERROR
  c_rule 2.2 (TOTAL_SUM $u lhs) EQ (TOTAL_SUM $u s1)]
```

Figure 5.3

In the above specification, the attribute **HOLD_SUM** should have been initialized to 0 in the initialization rule 2.1 and not to the character 'o'. Such kind of type errors will not be reported by **WAGE-ed**, but if a specification containing a type error is compiled, the line number of the right hand side interpreter definition whose associated semantic rule contains the type error will be reported by the Miranda compiler.

No Mutually Recursive Definitions

WAGE-ed requires that interpreter names used in the right hand side interpreter definition be previously defined. An implication of this constraint is that mutually recursive definitions have to be treated in a special way. If a W/AGE specification containing mutually recursive definition (either direct or indirect) is entered using **WAGE-ed**, then interpreter definitions involved in mutual recursion will not be annotated with appropriate warning messages. The example specification listed below, which contains mutual recursion, will help to illustrate this limitation.

```
------------------------------------------------------------
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>

attribute
    LITERAL_VAL terminal
    | VAL num
    | ADD_CONST num

reserved_words
  []

special_symbols
  ['+','(',')']

a_number
  literal INT_TERM

plus
  uninterpreted (SPECIAL_SYMBOL_TERM "+")

open_bracket
  uninterpreted (SPECIAL_SYMBOL_TERM "(")

close_bracket
  uninterpreted (SPECIAL_SYMBOL_TERM ")")
factor
```

Figure 5.4 (Continued . . .)

```
factor
  structure ( s1 open_bracket ++ s2 term{UNDEFINED} ++
              s3 close_bracket )
  [c_rule 1.1 (VAL $u lhs) EQ (VAL{ERROR} $u  s2) ]
  $excl_orelse
  structure ( s1 a_number )
  [a_rule 1.2 (VAL $u lhs) EQ conv_to_val[LITERAL_VAL $u  s1] ]
term
  structure ( s1 factor ++ s2 plus ++ s3 term )
  [c_rule 2.1 (ADD_CONST $d s1) EQ (ADD_CONST $d lhs) ,
  c_rule 2.2 (ADD_CONST $d s3) EQ (ADD_CONST $d lhs) ,
  a_rule 2.3 (VAL $u lhs) EQ do_sum[VAL $u  s1,VAL $u  s3] ]
  $excl_orelse
  structure ( s1 factor )
  [c_rule 2.4 (ADD_CONST $d s1) EQ (ADD_CONST $d lhs) ,
  c_rule 2.5 (VAL $u lhs) EQ (VAL $u  s1) ]
expr
  structure ( s1 term )
  [i_rule 3.1 (ADD_CONST $d s1) EQ (ADD_CONST 1) ,
  c_rule 3.2 (VAL $u lhs) EQ (VAL $u  s1) ]
-------------------------------------------------------------
```

Figure 5.4

The attribute grammar specification listed above was developed using **WAGE-ed** and saved as a text file. As one can observe, the interpreter definitions for **factor** and **term** are mutually-recursive. Because of mutual recursion, even though interpreter **term** has been defined, it is annotated with the warning message **{UNDEFINED}** in the definition of interpreter **factor**. Second, in the semantic rule 1.1, the attribute **VAL** is annotated with the warning message **{ERROR}** even though it is being passed up **s2**, *i.e.*, interpreter **term** in the semantic rules 2.3 and 2.5. And finally, the interpreter name **term** appearing on the right hand side in the first interpreter definition option for **factor**, should have been annotated with the warning message **{MISSING INH. ATTR. DEFN.}** because in the associated semantic rule we have not defined a semantic rule to pass down the inherited

attribute **ADD_CONST** to s2, *i.e.*, interpreter **term**, which is required.

§ 5.5 Support for Left-Recursive Interpreter Definitions in WAGE-ed

The *guarded* version of **WAGE-ed** supports left-recursive interpreter definitions. The behavior and working of guarded editors are exactly similar to unguarded editors, except that guarded editors have two additional types of interpreters, *viz.*:

- **recognised**: this interpreter type is similar to **structured** interpreter type, except that the productions do no have semantic rules associated with them. Also, interpreters defined as **recognised** should be non-left-recursive. These interpreters are used as guards to recognize left-recursive interpreter definitions,

- **guarded**: these interpreters can have left-recursive interpreter productions with associated semantic rules. Interpreters defined as **guarded** interpreters are also similar to **structured** interpreters, except that they have *guards* (interpreters which must be defined as **recognised** interpreters and they must be defined before their use) which enables left-recursive interpreter definitions if and only if the application of the **recognised** interpreter, used as a guard, to an input string terminates.

§ 5.6 Implementation of WAGE-ed

WAGE-ed has been implemented using the Synthesizer Generator [48], which

is itself based on the concept of attribute grammar programming. This programming paradigm provides a powerful mechanism for specifying how widely separated parts of a tree are constrained in the context provided by the rest of the tree.

The reader should note that this section is concerned only with the implementation of **WAGE-ed** and not its use. The fact that **WAGE-ed** is implemented in an attribute grammar programming language may be a little confusing at first and the reader should be careful not to confuse what is discussed in this section with the use of **WAGE-ed** as a tool to support attribute grammar programming.

One of the components of the specification for **WAGE-ed** are the *attribute equations* that express context-sensitive constraints. As an object is edited with **WAGE-ed**, it is represented as a derivation tree that is consistently attributed in accordance with the grammar's attribute equation. A tree is consistently attributed when the value of each attribute instance in the tree is equal to the value of the appropriate attribute definition function applied to the value of the neighboring attribute instances in the tree. For example, when a tree is modified by, say, some editing operations then some of the attributes may no longer have consistent values. Incremental analysis is performed by updating attribute values throughout the tree in response to the modification. The dependency relationship between attributes defined in the specification for **WAGE-ed** is used to reestablish consistent values.

In **WAGE-ed**, various constraints are expressed by having certain attributes in the specification indicate whether the constraint is satisfied. In addition to the attribute grammar component, the specification for **WAGE-ed** consists of *unparsing* rules that determines how objects are displayed on the screen. Attributes used in the unparsing specification cause the screen to be annotated with values of attribute instances. Basically, the attribute that indicate satisfaction or violation of the specified constraints are used to annotate the display to indicate the presence or absence of errors. If an editing operation modifies an object in such a way that formerly satisfied constraints are now violated or vice-versa, the attribute that indicate satisfaction of constraints will receive new values. The changed values of the attributes are used to provide feedback to the user about new errors introduced and/or old errors corrected.

The specification for **WAGE-ed** is specified using **SSL** (Synthesizer Specification Language), the language in which editors are specified. Each SSL specification consists of various *declarations*.

Abstract Syntax Declarations

The main component of the specification for **WAGE-ed** is the definition of W/AGE's abstract syntax, given as a set of grammar rules. The grammar rules are productions of a context-free grammar. The effect of each editing modification

is to change the underlying abstract syntax tree. The abstract syntax of W/AGE consists of a collection of productions of the form.

x_0: op(x_1 x_2 . . . x_k)

where **op** is an operator name and each x_i is a nonterminal of the grammar.

Attribute and Attribute Equations Declarations

The specification for **WAGE-ed** also consists of *attribute declarations* that associate attributes with nonterminals and productions. The attribute equations are used to define the value of attributes in terms of other attributes that occur in the production. If the value of an attribute is a function of other attributes that occur in the production, then these functions appear in *function declarations*. When the program is being edited, the underlying derivation tree is fully and consistently attributed, all attributes of all nonterminals and productions are given values. Attribute values are updated automatically as objects are modified.

The attribute equations make use of synthesized and inherited attributes to make static inferences about the objects being edited. The warning messages that annotate the program as it is edited arise from static inferences about whether the program violates various constraints.

Unparsing Declarations

The display representation of the derivation trees derived from nonterminals of the grammar is defined by *unparsing declarations*. These declarations determine

how various objects in **WAGE-ed** are displayed on the screen. The display representation, in some cases, is influenced by the value of attributes. Therefore, after each editing operation, attribute values are reevaluated and only then the objects are displayed on the screen.

Concrete Input Syntax Declarations

In **WAGE-ed** objects are modified by three kinds of editing operations *viz.* text editing, transformation operations and system commands. System commands are same in all editors generated with the Synthesizer Generator, but text editing and transformation operations are defined in the specification of **WAGE-ed**

Whenever a placeholder is refined by text, it is parsed and translated to an abstract form which is determined by an input syntax. In **WAGE-ed**, *parsing declarations* define the productions of a grammar to be used for parsing the text. Attribute equations associated with the productions of the input syntax define the translation of text to abstract syntax.

The *entry declarations* defined in the specification of **WAGE-ed** are used to establish a correspondence between the abstract syntax of objects editable as text and the subsets of the input syntax, because at any given moment only a subset of the input syntax is recognized. Each well-formed text in a given context determines a parse tree with respect to the input syntax. This parse tree is translated into an abstract syntax tree with attribution.

Transformation Declarations

The *transformation declarations* defined in the specification for **WAGE-ed** specify how to restructure a selected object, when the component located at the selection matches a given structural pattern.

A transformation determines a replacement value for the selected subterm as a function of its current value. At any given moment during editing, a transformation is either enabled or disabled depending on whether or not its pattern matches the value of the selection. Transformations which are enabled are listed in the help pane, they can be invoked either by clicking on them in the help pane or selecting them from the menu. Transformations maintain the syntactical integrity of the program, because their definitions are type checked during compilation.

The complete code for **WAGE-ed** is listed in Appendix 2.

Chapter 6 Conclusion

In this section I recapitulate the result of my thesis and give directions for future research work.

§ 6.1 Assessing the value of WAGE-ed

The driving force behind the construction of the editor is that it should facilitate in the construction of syntactically correct W/AGE program, allow programs to be constructed at a higher level of abstraction, perform automatic indentation of the program, insert punctuation symbols automatically, and provide the user with feedback about missing attribute equations.

In order to support my thesis, I have:

• Built a structure editor, **WAGE-ed**, for W/AGE — an environment for constructing executable specification of attribute grammars,

• Surveyed users of **WAGE-ed** and summarized their experience with it.

The main conclusions which I draw after implementing the editor are:

• **WAGE-ed** prohibits *all* errors which emanate as a result of misspelled W/AGE keywords, missing punctuation symbols, incorrect indentation of the program, undefined interpreters, *etc.* Users of **WAGE-ed** are relieved of the frustration and time wasted in debugging these kind of errors, because the editor simply prohibits them.

• **WAGE-ed** facilitates in the construction of executable specification of attribute grammars at a higher-level of abstraction, without having to bother about the fine implementation details.

• **WAGE-ed** signals, with appropriate warning message, the use of undeclared attributes, and reference to synthesized or inherited attribute values whose associated attribute equations are undefined.

• **WAGE-ed** is easy to use and learn.

• **WAGE-ed**, to some extent, helps users learn the concept of attribute grammar programming.

The last two points are the analysis of the survey which I conducted after implementing the editor. The main idea of surveying the users of **WAGE-ed** was to obtain answers for the following questions:

• Are there any disadvantages, such as the time to learn how to use the editor, of using a specialized editor which might offset the facilities provided by it ?

• Does it help the user in learning attribute grammar programming ?

The users of **WAGE-ed**, who were surveyed, were undergraduate students (a mix of second and third year students) who took the 60–214 Compiler course offered by the School of Computer Science. The students used **WAGE-ed** for their assignments and project.

• A total of 43 students (1 Master's , 20 Third Year, and 22 Second Year

students) were surveyed.

- All of the students had experience with more than one programming language, *viz.*, C and Turing.

- None of them had any prior experience using a structure editor.

The graph shown below illustrates the time taken by students to learn using **WAGE-ed** without any assistance, *i.e.*, without any help from the GA or the course instructor, except for reference to the user manual. Out of a total of 43 students, 37 responded.

ANALYSIS OF SURVEY

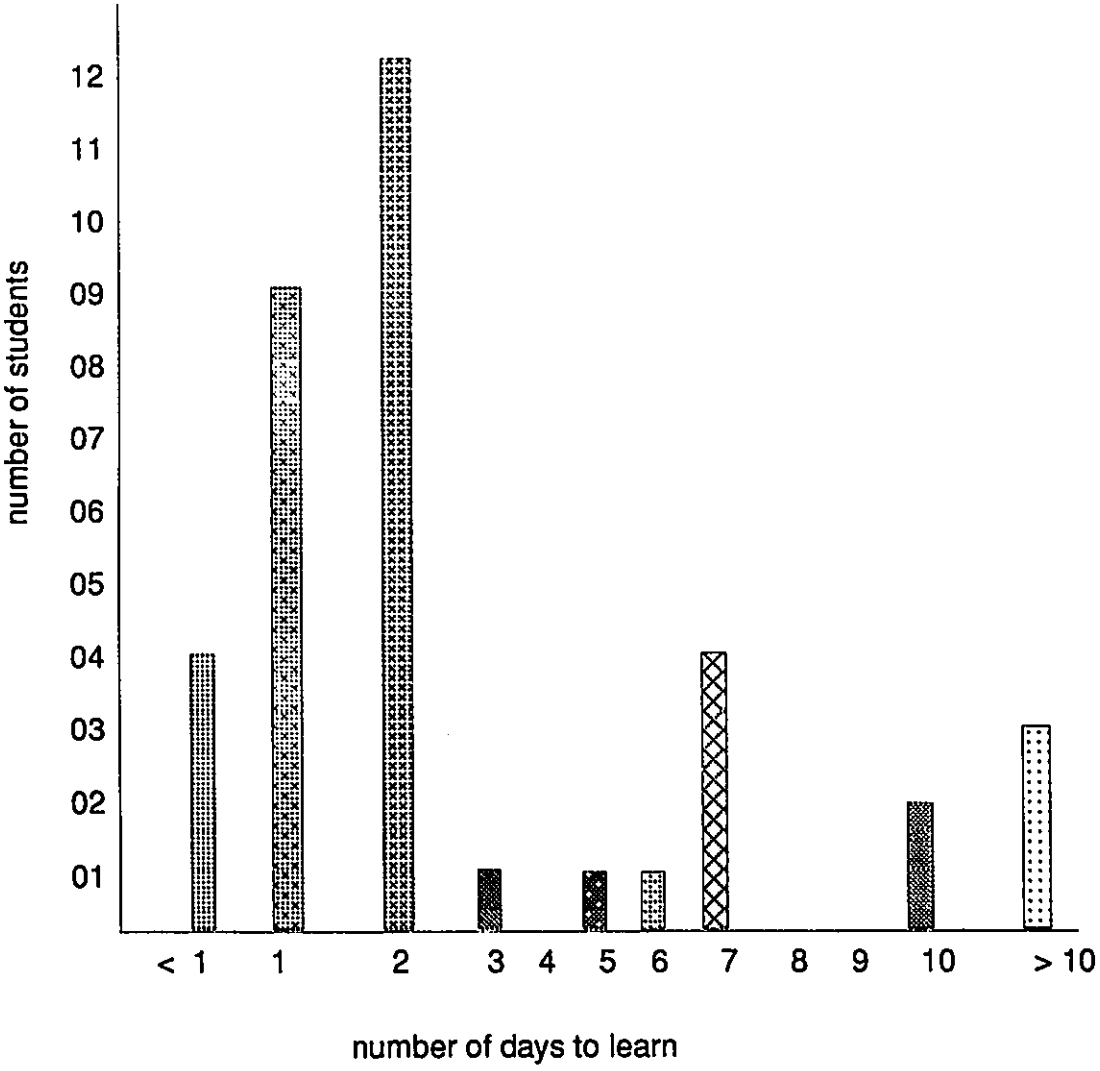

Figure 6.1

As seen in the graph above, most of the students took less than a week to

learn to use **WAGE-ed**, the average being around 4 days.

Almost 65% of the students surveyed responded that **WAGE-ed** helped them learn the attribute grammar programming paradigm.

The results of the survey and feedback from users of **WAGE-ed** has confirmed my belief that a structure editor is useful to novice users who are unfamiliar with a particular programming paradigm or language and also to experienced programmers who are switching from one programming language to another. Because it helps them to become familiar to the new environment in a relatively short period of time and acts like a guide overlooking their shoulder for various kinds of errors. As a result, a beginner is not intimidated by unfriendly error messages issued by the compiler because they are taken care of during the editing phase itself and at the same time learns to write programs in the new language relatively fast.

In our case, I believe that **WAGE-ed** will:

• Help users unfamiliar with attribute grammar programming paradigm, learn it through the various warning messages issued by **WAGE-ed** due to incorrect use of synthesized and/or inherited attributes.

• Considerably reduce the time taken to develop a W/AGE program, free of syntax errors and errors in the use of attributes, by saving time in learning the syntax and layout rules of W/AGE and time spent debugging errors.

• Help users unfamiliar with W/AGE become familiar with it. Because **WAGE-ed**: automatically handles insertion of various keywords, punctuation symbols, layout of the program; provides predefined templates for various types of interpreter definitions, reserved words, special symbols, *etc.*, it can be used as a manual to learn W/AGE.

From the facilities supported by **WAGE-ed** in the construction of executable specification of attribute grammars and the results of the survey, I conclude that **WAGE-ed** is of value in the construction of executable specification of attribute grammars and at the same time is easy to use and learn.

§ 6.2 Prospects for Future Work

Interactive Programming Environment for W/AGE

WAGE-ed currently facilitates the construction of syntactically correct attribute grammar specifications and checks for missing attribute declarations and attribute equations.

WAGE-ed does not support structure editing facility for the programming language Miranda. It would have been possible to create a structure editor for Miranda that would check for syntax, semantic and type errors, using the Synthesizer Generator if more time had been available. It would be challenging to develop an interactive programming environment for W/AGE, which facilitates

the development of attribute grammar specifications, checks Miranda functions for syntax and semantic violations and performs type checking.

Another important feature to be included would be interactive testing and debugging of attribute grammar specifications. It would be desirable to initiate the execution of the program at any stage of execution, with no delay for compilation. It would be interesting to investigate whether or not it is possible to produce and maintain an executable code in Miranda as attribute grammar specifications are developed using **WAGE-ed.**

Developing Benchmarks for Interactive Program Development Environments

There has been absolutely no work done in developing benchmarks for interactive programming environments supporting structure editing facility. A programming environment for W/AGE with the above mentioned features could be used as a testbed to develop benchmarks for interactive program development environments. These benchmarks could be used to test the versatility and features of current and future interactive program development environments and check their effectiveness in increasing the productivity of the programmer.

Handling Mutually Recursive Definitions

WAGE-ed at present does not have any facility for entering mutually recursive definitions. It would be interesting to investigate whether or not it is possible to extend **WAGE-ed** to support mutually recursive definitions.

Writing Concrete Input Syntax for the Entire W/AGE Language

If a W/AGE program developed using **WAGE-ed** is saved as a text file then it is not possible to read that file back into **WAGE-ed** for further editing.

It is possible to do this by extending the **WAGE-ed** specification to include concrete input syntax (refer to section 5.7) for the entire W/AGE language.

Evaluating Various Aspects of WAGE-ed

Due to unavailability of time, it was not possible to perform experiments to evaluate various aspects of **WAGE-ed**, namely:

- Ease with which programs being developed using **WAGE-ed** could be modified, *viz.*, inserting and deleting placeholders and text by making appropriate selections from the pop-up menu.

- Ease of performing cut and paste operations. These operations might be prohibited, by **WAGE-ed**, at some locations in the program which the user might feel is inappropriate. But as I have mentioned no rigorous experiments were conducted to check for all possible locations where these operations would prove to be necessary and useful. If all such probable locations are found, it is just a matter of modifying unparsing properties of appropriate nodes in the abstract syntax tree.

• Suitability of refining certain placeholders by entering text from the keyboard, rather than transforming them by making selections from choices listed in the help pane.

BIBLIOGRAPHY

[1] A.V. Aho, J.D. Ullman, and R. Sethi. *Compilers-Principles, Techniques and Tools*. Addison-Wesley, 1986.

[2] C.N. Alberga, A.L. Brown, G.B. Leeman Jr., M. Mikelsons, and M.N. Wegman. A Program Development Tool. In *Conference Record of the Eigth ACM Symposium on Principles of Programming Languages*, Williamsburg, Virginia, January 1981.

[3] M.B. Albizuri-Romero. GRASE-A Graphical Syntax Directed Editor for Structured Programming. *ACM SIGPLAN*, 19(2), February 1984.

[4] L. Allison. Syntax Directed Program Editing. *Software-Practice and Experience*, 13, 1983.

[5] F. Arefi, C.E. Hughes, and D.A. Workman. Automatically Generating Visual Syntax-Directed Editors. *Communications of the ACM*, 1990.

[6] L.V. Atkinson, J.J. McGregor, and S.D. North. Context Sensitive Editing as as Approach to Incremental Compilation. *The Computer Journal*, 24(3), 1981.

[7] D.J. Bagert and D.K. Friesen. A Multi-Language Syntax-Directed Editor. *ACM*, 1987.

[8] R. Bahlke and G. Snelting. Context-Sensitive Editing with PSG Environment. In *Proceedings of the International Workshop on Advanced Programming Environments : Lecture Notes in Computer Science*. Springer-Verlag, New York, June 1986.

[9] R. Bahlke and G. Snelting. The PSG System : From Formal Language Definitions To Interactive Programming Environments. *ACM Transactions on Programming Languages and Systems*, 8(4), October 1986.

[10]R.A. Ballance, S.L. Graham, and M.L. Van De Vanter. The Pan Language-Based Editing System. *ACM Transactions on Software Engineering and Methodology*, 1(1), January 1992.

[11]D.R. Barstow. A Display-Oriented Editor for INTERLISP. In *Proceedings of IJCAI-81*, August 1981.

[12]B.A. Bottos and C.M.R. Kintala. Generation of Syntax-Directed Editors With Text-Oriented Features. *The Bell Systems Technical Journal*, 62(10), December 1983.

[13]F.J. Budinsky, R.C. Holt, and S.G. Zaky. SRE : A Syntax Recognizing Editor. *Software-Practice and Experience*, 15(5), May 1985.

[14]M.F. Cowlishaw. LEXX- A Programmable Structured Editor. *IBM Journal of Research and Development*, 31(1), January 1987.

[15]S.A. Dart, R.J. Ellison, P.H. Feiler, and A.N. Habermann. Software Development Environments. *IEEE Software Engineering*, 1987.

[16]V. Donzeau-Gouge, G. Huet, G. Kahn, and B. Lang. *Programming Environments Based on Structured Editors : The MENTOR Experience*. McGraw-Hill, New York, 1984.

[17]V. Donzeau-Gouge, G. Huet, G. Kahn, B. Lang, and J.J. Levy. A Structure Oriented Program Editor : A First Step Towards Computer Assisted Programming. In *International Computing Symposium*. North Holland Publishing Company, 1975.

[18]L.R. Dykes and R.D. Cameron. Towards High-Level Editing in Syntax-Based Editors. *Software Engineering Journal*, July 1990.

[19]G. Engels, M. Nagl, and W. Schaefer. On the Structure of Structure-Oriented Editors for Different Applications. In *Proceedings of ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environment*, volume 22, 1987.

[20]C.N. Fischer, G.F. Johnson, J. Mauney, Anil Pal, and D.L. Stock. The Poe Language-Based Editor Project. In *ACM SIGSOFT SIGPLAN Symposium on Practical Software Development Environments*, Pittsburgh, Pennsylvania, April 1984.

[21]C. Fraser. Syntax-Directed Editing of General Data Structures. In *ACM SIGPLAN-SIGOA Symposium on Text Manipulation*, Portland, Oregon, June 1981.

[22]P. Fritzson. Preliminary Experience from the DICE system, a Distributed Incremental Compiling Environment. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, volume 19, Pittsburgh, Pennsylvania, May 1984.

[23]R. Frost. Constructing Programs as Executable Attribute Grammars. *The Computer Journal*, 35(4), 1992.

[24]R. Frost. Guarded Attribute Grammars. *Software Practice and Experience*, Inprint (May 1993).

[25]R. Frost. *Lecture Notes in Computer Science*, chapter Supporting the Attribute Grammar Programming Paradigm in a Lazy Functional Programming Language. Springer-Verlag, New York,, Inprint (May 1993).

[26]E. Gansner, J.R. Horgan, D.J. Moore, P.T. Surko, and D.E. Swartout. SYNED-A Language-Based Editor for an Interactive Programming Environment. In *Digest of Papers, Spring Compcon 83, Twenty-Sixth IEEE Computer Society International Conference*, February 1983.

[27]D.B. Garlan and P.L. Miller. GNOME : An Introductory Programming Environment Based on a Family of Structure Editors. *ACM*, 1984.

[28]K. Halewood and M.R. Woodward. NSEDIT : A Syntax-Directed Editor and Testing Tool Based on Nassi-Shneiderman Charts. *Software-Practice and Experience*, 18(10), October 1988.

[29]W.J. Hansen. User Engineering Principles for Interactive Systems. In *AFIP FJCC*, volume 39, 1971.

[30]F.C. Heeman, S. van Egmond, and J.C. van Vliet. INFORM : An Interactive Syntax Directed Formulae Editor. *The Journal of System and Software*, 9, 1989.

[31]J. Heyman and W.M. Lively. Syntax-Directed Editing Revisited. *ACM SIGSOFT Software Engineering Notes*, 10(3), July 1985.

[32]N. Holsti. A Session Editor with Incremental Execution Functions. *Software-Practice and Experience*, 19(4), April 1989.

[33]J.R. Horgan and D.J. Moore. Techniques for Improving Language-Based Editors. In *ACM SIGSOFT SIGPLAN Symposium for Practical Software Development Environments*, Pittsburgh, Pennsylvania, April 1984.

[34]C.R. Jesshope, M.J. Crawley, and G.L. Lovegrove. An Intelligent Pascal Editor for a Graphical Oriented Workstation. *Software-Practice and Experience*, 15(11), November 1985.

[35]G.E. Kaiser, P.H. Feiler, F. Jalilli, and J.H. Schlichter. A Retrospective on DOSE : An Interpretive Approach to Structure Editor Generation. *Software-Practice and Experience*, 18(8), August 1988.

[36]G.E. Kaiser, S.M. Kapian, and J. Micallef. Multiuser, Distributed Language-Based Environments. *IEEE Software*, 1987.

[37]B. Lang. On the usefulness of syntax directed editors. In *Lecture Notes in Computer Science*. Springer-Verlag, New York, 1986.

[38]J.W. Lewis and D.F. Porges. ALBE/P : A Language-Based Editor for Pascal. In *Proceedings of the Eigth Texas Conference on Computing Systems*, November 1979.

[39]T.F. Lunney and R.H. Perrott. Syntax-Directed Editing. *Software Engineering Journal*, March 1988.

[40]R. Medina-Mora and P.H. Feiler. An Incremental Programming Environment. *IEEE Transactions on Software Engineering*, 7(5), September 1981.

[41]J.M. Morris and M.D. Schwartz. The Design of a Language-Directed Editor for Block-Structured Languages. In *Proceedings of the ACM SIGPLAN-SIGOA Symposium on Text Manipulation*, volume 16, Portland, Oregon, June 1981.

[42]C. Nicol, M.K. Crowe, M.E. Corr, J.W. Oram, and D.G. Jenkins. IDEA : An Incremental Development Environment for Ada. *Software Engineering Journal*, November 1987.

[43]D. Notkin. Structured User Environments. In *Proceedings of the Associated Simula Users Workshop on Program Development Tools*, November 1982.

[44]S. Ola. *Lecture Notes in Computer Science*, volume 244, chapter Editing Large Programs Using a Structure-Oriented Editor. Springer-Verlag, New York, 1986.

[45]J. Parker. Towards More Intelligent Programming Environments. *ACM SIGSOFT Software Engineering Notes*, 10(3), July 1985.

[46]R.H. Perrott and T.F. Lunney. A Syntax-Directed Integrated Programming Environment for Developing SIMD Supercomputer Software. *Software-Practice and Experience*, 21(3), March 1991.

[47]S.P. Reiss. Graphical Program Development with PECAN Program Development Systems. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, Pittsburgh, Pennsylvania, April 1984.

[48]T. Reps and T. Teitelbaum. The Cornell Program Synthesizer : A Syntax-Directed Programming Environment. *Communications of the ACM*, 24(9), September 1981.

[49]T. Reps and T. Teitelbaum. The Synthesizer Generator. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, volume 19, Pittsburgh, Pennsylvania, May 1984.

[50]T. Reps and T. Teitelbaum. Language Processing in Program Editors. *IEEE Software Engineering*, November 1987.

[51]T. Reps, T. Teitelbaum, and S. Horowitz. The Why and Wherefore of the Cornell Program Synthesizer. In *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, volume 16, June 1981.

[52]U. Shani. Should Program Editors Not Abandon Text Oriented Commands ? *ACM SIGPLAN Notices*, 18(1), January 1983.

[53]E. Shapiro, G. Collins, L. Johnson, and J. Ruttenberg. PASES : A Programming Environment for PASCAL. Technical report, Yale University, New Haven, Connecticut, 1980.

[54]R.M. Stallman. EMACS : The Extensible, Customizable, Self-Documenting Display Editor. In *Proceedings of the ACM SIGPLAN-SIGOA Symposium on Text Manipulation*, volume 16, Portland, Oregon, June 1981.

[55]W. Teitelman. Automated Programmering : The Programmer's Assistant. In *Proceedings of the Fall Joint Computer Conference, AFIPS Proceedings*, 1972.

[56]F. Wadia. Survey on Structure Editors. Technical report, School of Computer Science, University of Windsor, August 1992.

[57]R.C. Waters. Program Editors Should Not Abandon Text Oriented Commands. *ACM SIGPLAN Notices*, 17(7), July 1982.

[58]J. Wilander. An Interactive Programming System for Pascal. *BIT*, 20. 1980.

[59]S.R. Wood. Z-The 95% Program Editor. In *Proceedings of the ACM SIGPLAN-SIGOA Symposium on Text Manipulation*, volume 16, Portland, Oregon, June 1981.

[60]M.V. Zelkowitz. A Small Contribution to Editing with a Syntax Directed Editor. *ACM SIGPLAN Notices*, 9(3), 1984.

APPENDIX 1
An Example Session with a Structure Editor

This section illustrates an editing session with a structure editor for a subset of Pascal-like programming language.

The language supports four kinds of statements, *viz.*, an assignment statement, an **if-then-else** statement, a **while** statement, and compound statement, *i.e.*, a sequence of statements enclosed between the keywords **begin** and **end**. The body of the **while** and **if-then-else** statements can have only one statement (if a sequence of statements is required then they should be enclosed between the keywords **begin** and **end**). The language supports two kinds of data types, *viz.*, **integer** and **boolean**. The language requires that all variables be declared before their use and the language is strongly typed, *i.e.*, an **integer** value cannot be used at a place which requires a **boolean** and vice-versa.

The editor used for the demonstration illustrates the kind of structure editor that can be built using the **Cornell Synthesizer Generator** [48] and the features that can be incorporated into it.

When the editor is initially loaded, it appears as shown in Figure 1.1.

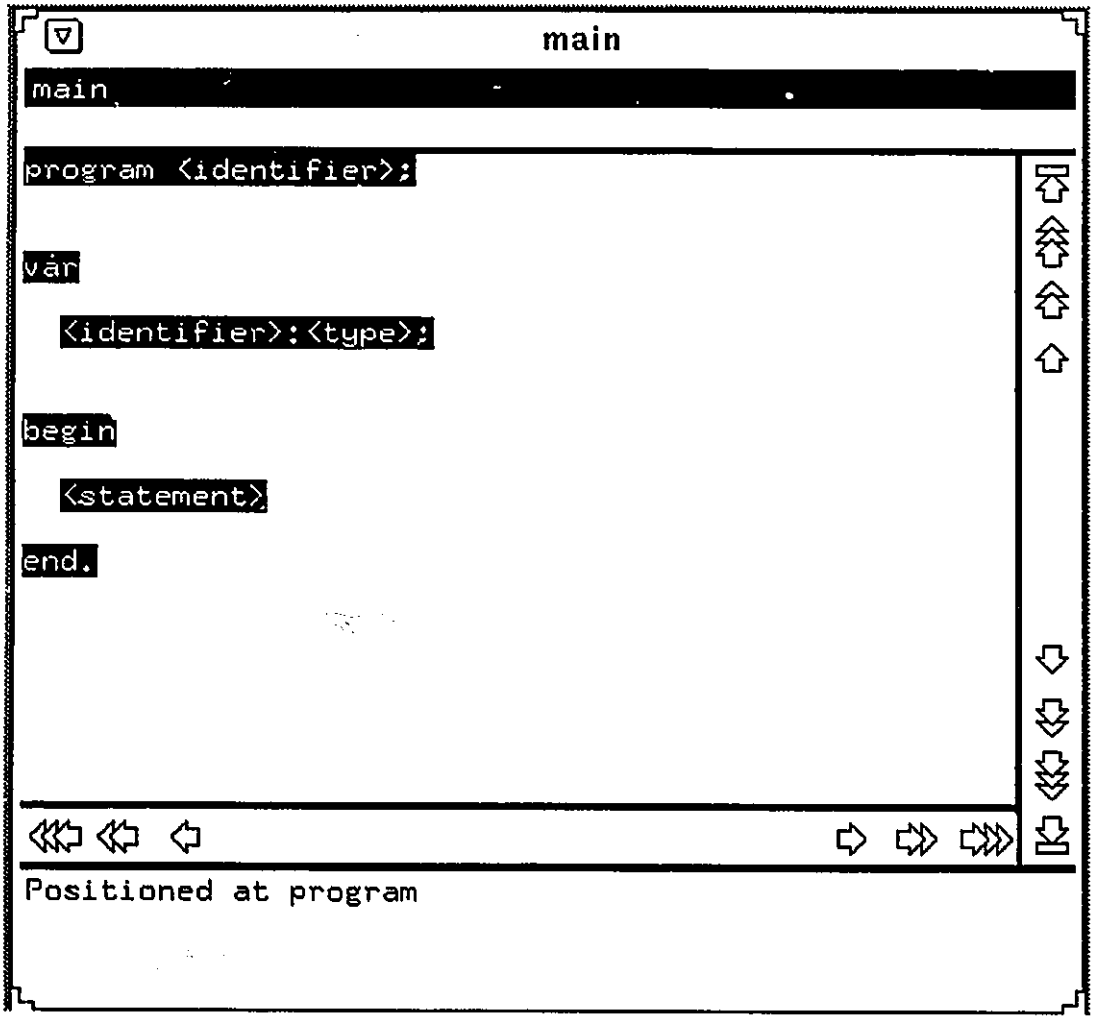

Figure 1.1

The screen, as shown in Figure 1.1, is divided into four regions:

• The topmost highlighted region, the *title bar*, containing the identifier **main**, indicates the name of the buffer whose contents are being currently edited. By default, this buffer is named **main**.

• The next region following the title bar displays system error messages. The editor ***will not*** allow any further edits before the error displayed in this region is corrected.

• The next region is the place where one enters the program, the *object pane*. It consists of text enclosed within angle brackets, known as *placeholders*, these are the locations which need to be refined or locations where additional text can be inserted. In order to insert text at a particular placeholder, one has to select the placeholder by placing the mouse arrow on the placeholder to be refined and clicking the leftmost mouse button. On doing so the placeholder is selected and appears in reverse video. If there are any *transformations* associated with the placeholder they will be listed in the lowermost region of the window, the *help pane*. In such a case, the user can select one of the choices listed by placing the mouse arrow on one of them and clicking the leftmost mouse button. If there are no transformations listed, that means the user can enter text from the keyboard to refine the selected placeholder. Text not enclosed within angle brackets are language keywords and are ***immutable***.

• The lowermost region of the window, the *help pane*, displays the node or *phylum* in the abstract syntax tree which is the current selection and the legal transformations associated with it, if any.

The arrows which appear on the right hand side and lower half region of the

screen can be used to move the program text vertically and horizontally.

As seen in Figure 1.1, initially, the entire program template is selected. At this point one can choose to refine any placeholder in the template by selecting it. Suppose we choose to refine the statement placeholder, on doing so the screen appears as shown in Figure 1.2.

As we can see in Figure 1.2, the statement placeholder is now our current selection. Also, the legal transformations associated with the statement placeholder are listed in the help pane. In order to refine the statement placeholder we can select any one of the choices. Suppose we select **assign**. On doing so, the statement placeholder is transformed into one shown below:

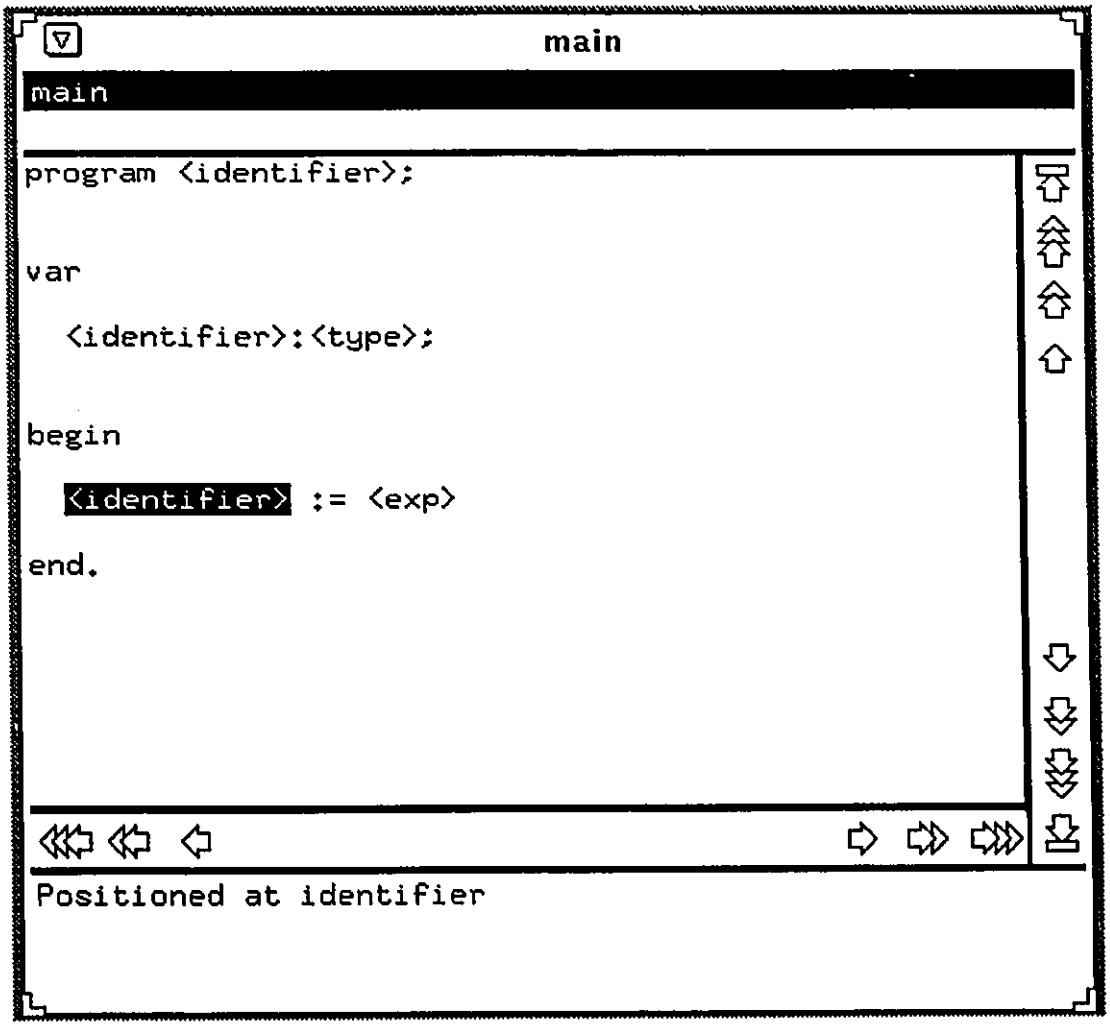

Figure 1.2

As seen in Figure 1.3, the statement placeholder has been transformed into an assignment statement consisting of an identifier placeholder and an expression placeholder. The identifier placeholder is our current selection, since there are no transformations associated with it we can enter any legal identifier (consisting

of alphanumeric characters, starting with an alphabet and can have embedded underscores).

Suppose, we key in an illegal identifier and then try to select the expression placeholder. But the system will not allow us to select the expression placeholder because we have entered an illegal identifier and a system error message will be displayed. We have to rectify our error in order to perform further edits. The sequence of performing above action is shown in Figure 1.4.

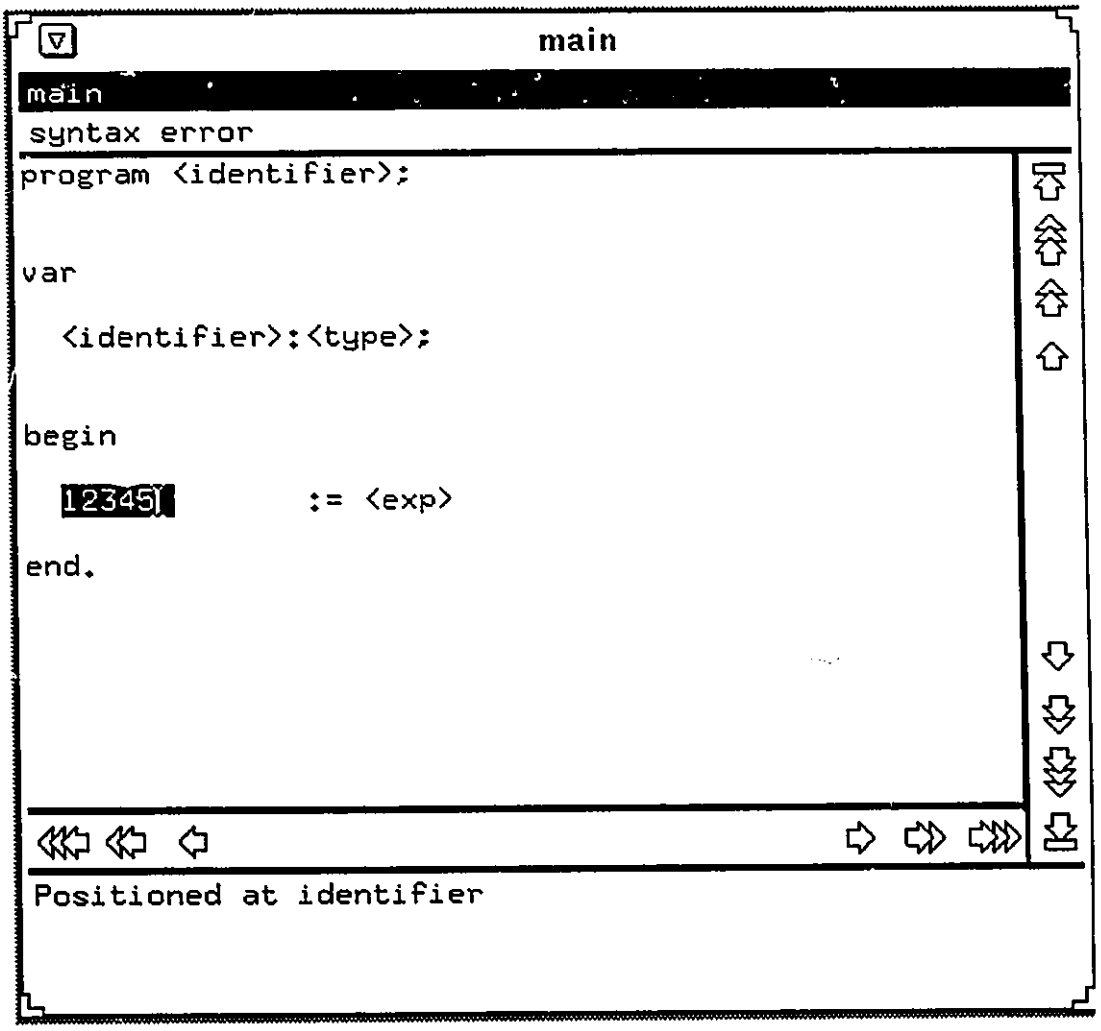

Figure 1.3

We can use the delete key to erase the illegal identifier and then retype a legal identifier. Next we select the expression placeholder, as seen in Figure 1.5.

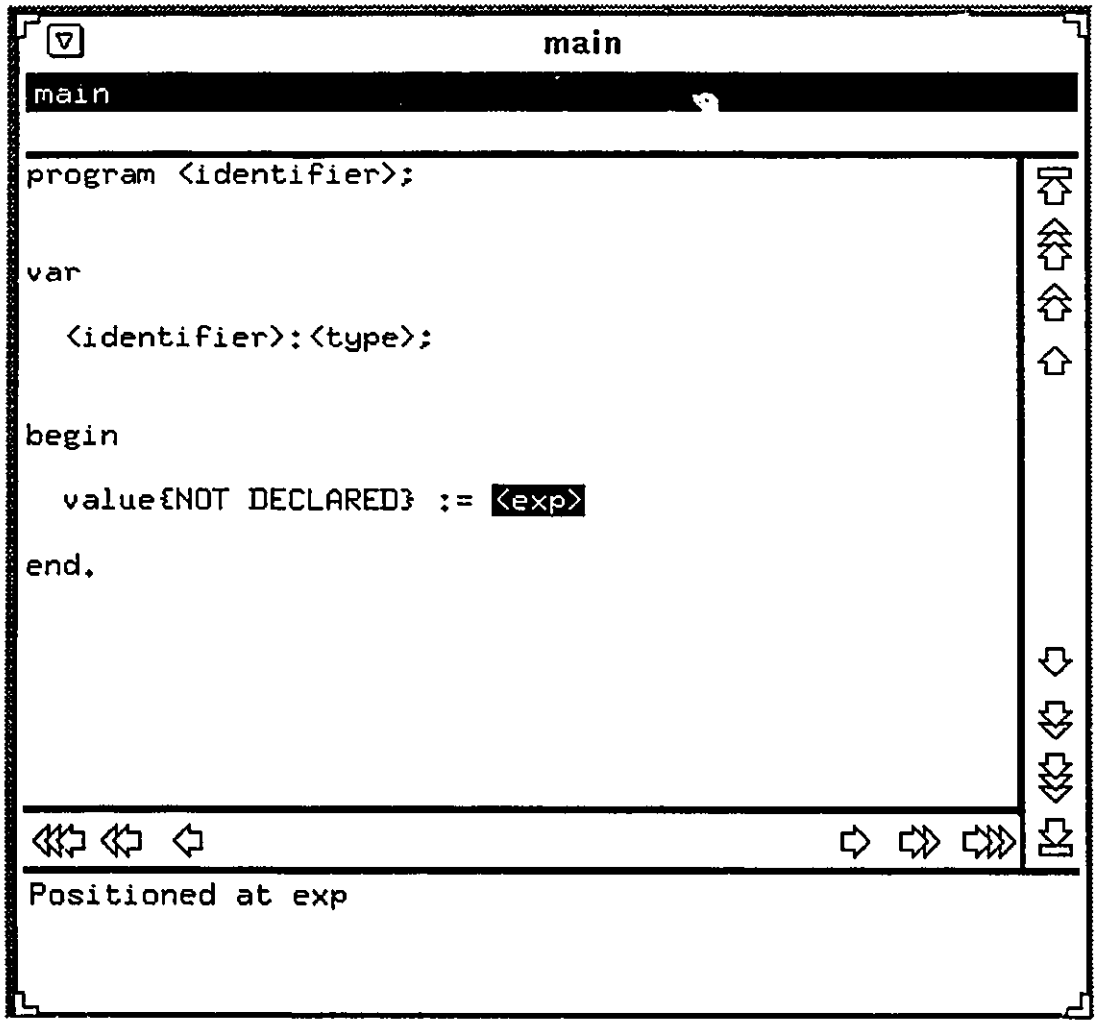

Figure 1.4

The system, before making any placeholder a current selection, makes sure whether or not the previous placeholder was transformed into a legal sentence. In the above figure, we entered an undefined legal identifier, the system responded with a warning message and made the expression placeholder the current selection.

A warning message need not be rectified as soon as it is introduced, but it serves to inform the user that an appropriate action must be taken in order for the program to compile correctly.

Next, we refine the expression placeholder by entering the text **value + 1** and hit the return key. After entering the return key, the system parses the previously entered text and takes appropriate action.

As exemplified by Figure 1.6 with the automatic insertion of the parentheses and semi-colon, a structure editor can be crafted to perform desired pretty-printing, indentation, display appropriate warning messages, *etc.* Now, let us define the type for the identifier **value.** In order to do so, we select the identifier placeholder present in the variable declaration section. First, we key in the identifier name then we select the type placeholder and choose **boolean** type for it and finally hit return. The system takes the actions as seen in Figure 1.7.

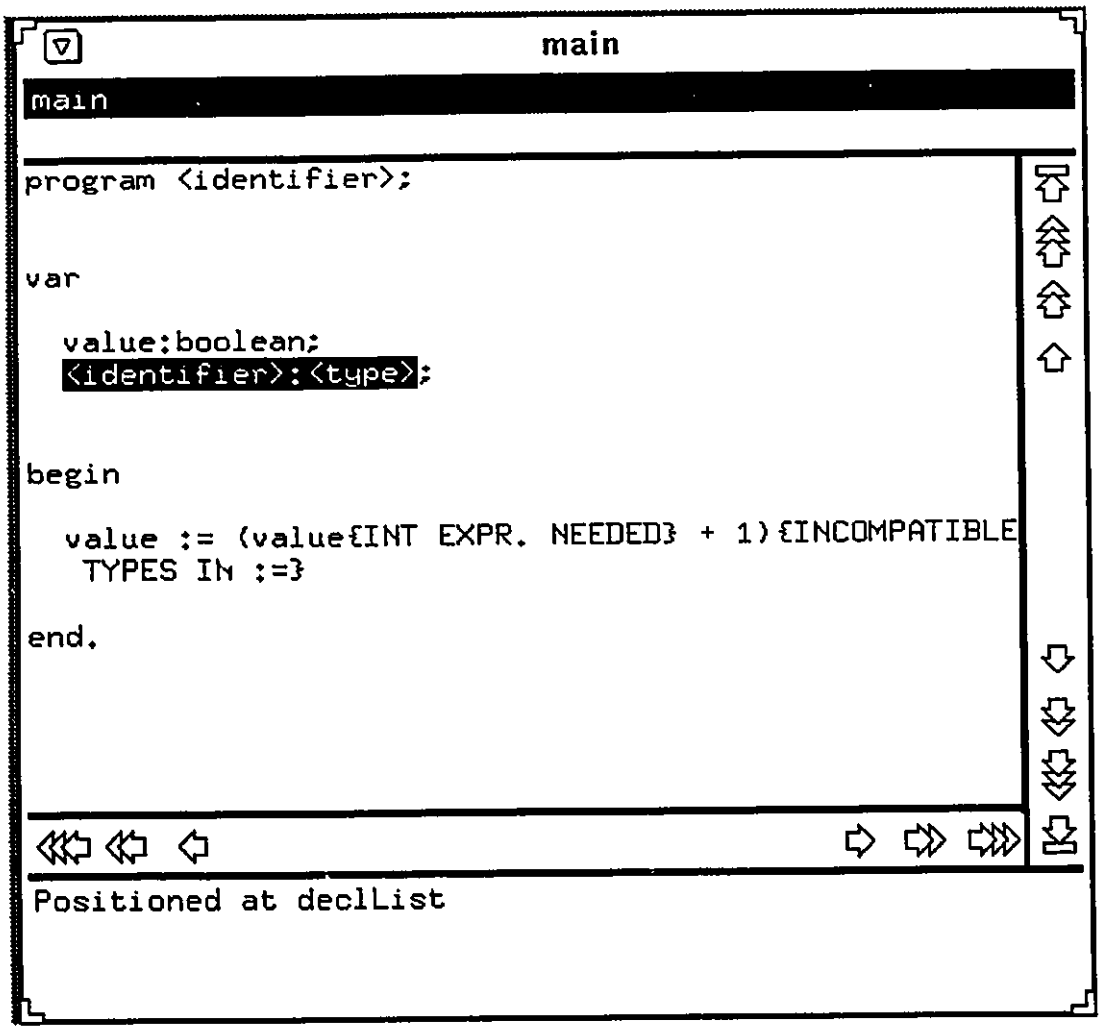

Figure 1.5

As seen in Figure 1.7, the editor has attributed the assignment statement with appropriate warning messages, because we have used a boolean variable in an expression which requires an integer. Also, the editor has introduced another

template for variable declaration. Next we introduce[4] a **while** statement as seen in Figure 1.8.

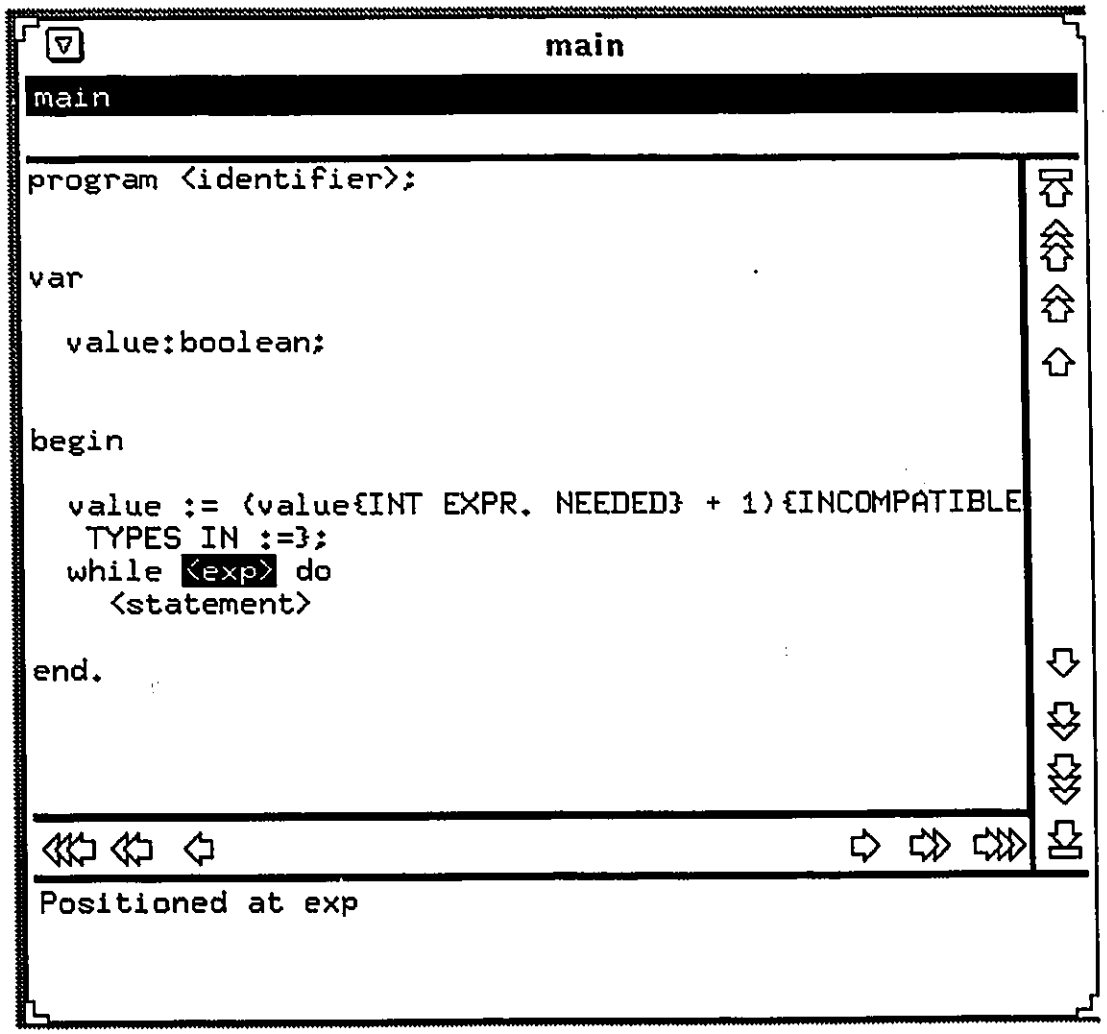

Figure 1.6

[4] The procedures to introduce placeholders, perform copy-paste-cut operations, walk through the abstract syntax tree *etc.* are explained in detail in Appendix C.

As we can see in the figure above, the body of the **while** statement has been appropriately indented. We enter, **(value <> 100)** and **value := value + 1**, for expression and statement placeholder respectively and hit return. When each of these texts is entered it is parsed to check whether it is syntactically and semantically correct. After performing the above actions, the screen appears as seen in Figure 1.9.

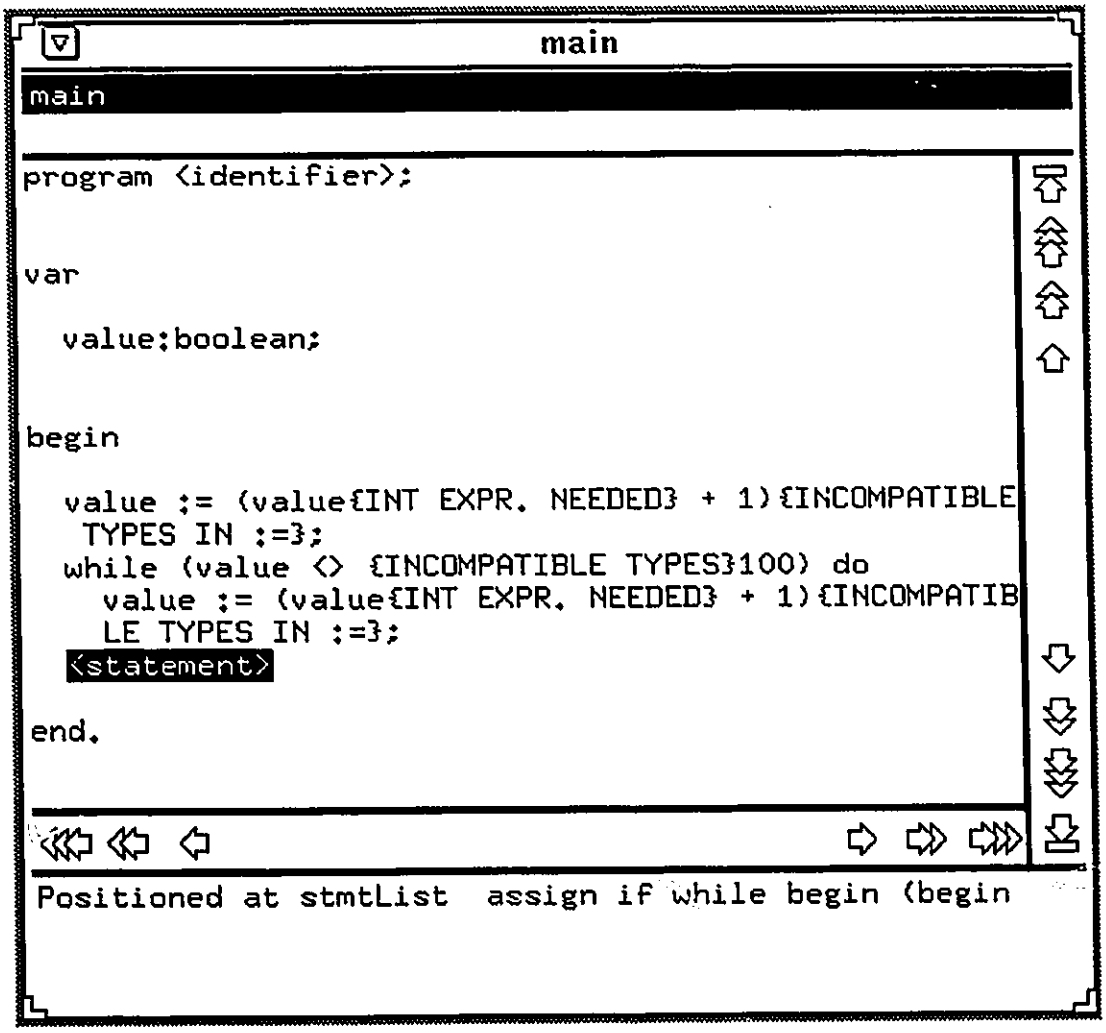

Figure 1.7

As seen in Figure 1.9, the next statement placeholder is outside the scope of the **while** statement. Suppose we want to define multiple statements for the **while** statement, we can do this by selecting only the body of the **while** statement (this can be done by placing the mouse arrow on := symbol and clicking the left

mouse button) as seen in Figure 1.10.

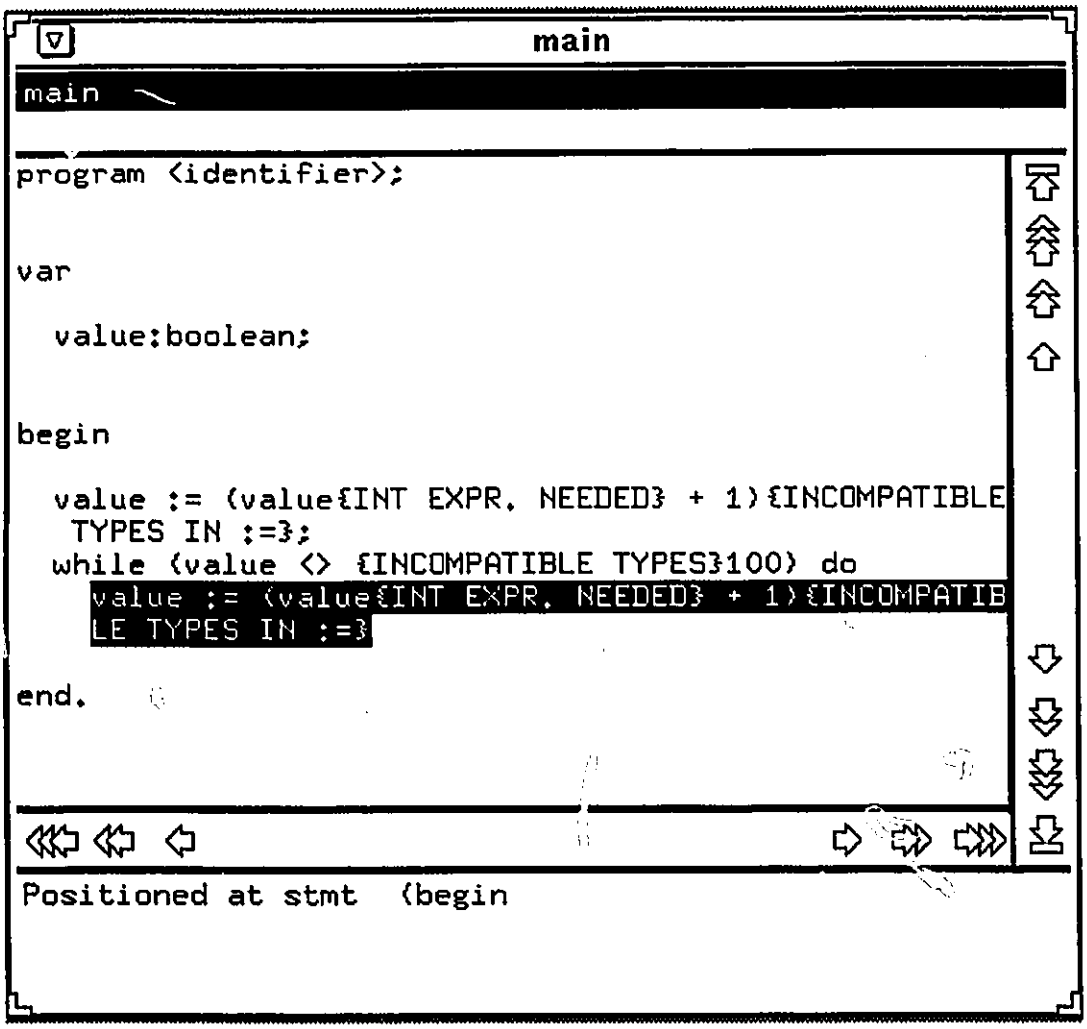

Figure 1.8

As we can see in Figure 1.10, there is one transformation associated with the highlighted statement, namely **(begin**. If we click on this choice the entire body

of the **while** statement will be enclosed within the keywords **begin** — **end** and then we can define multiple statements. See Figure 1.11.

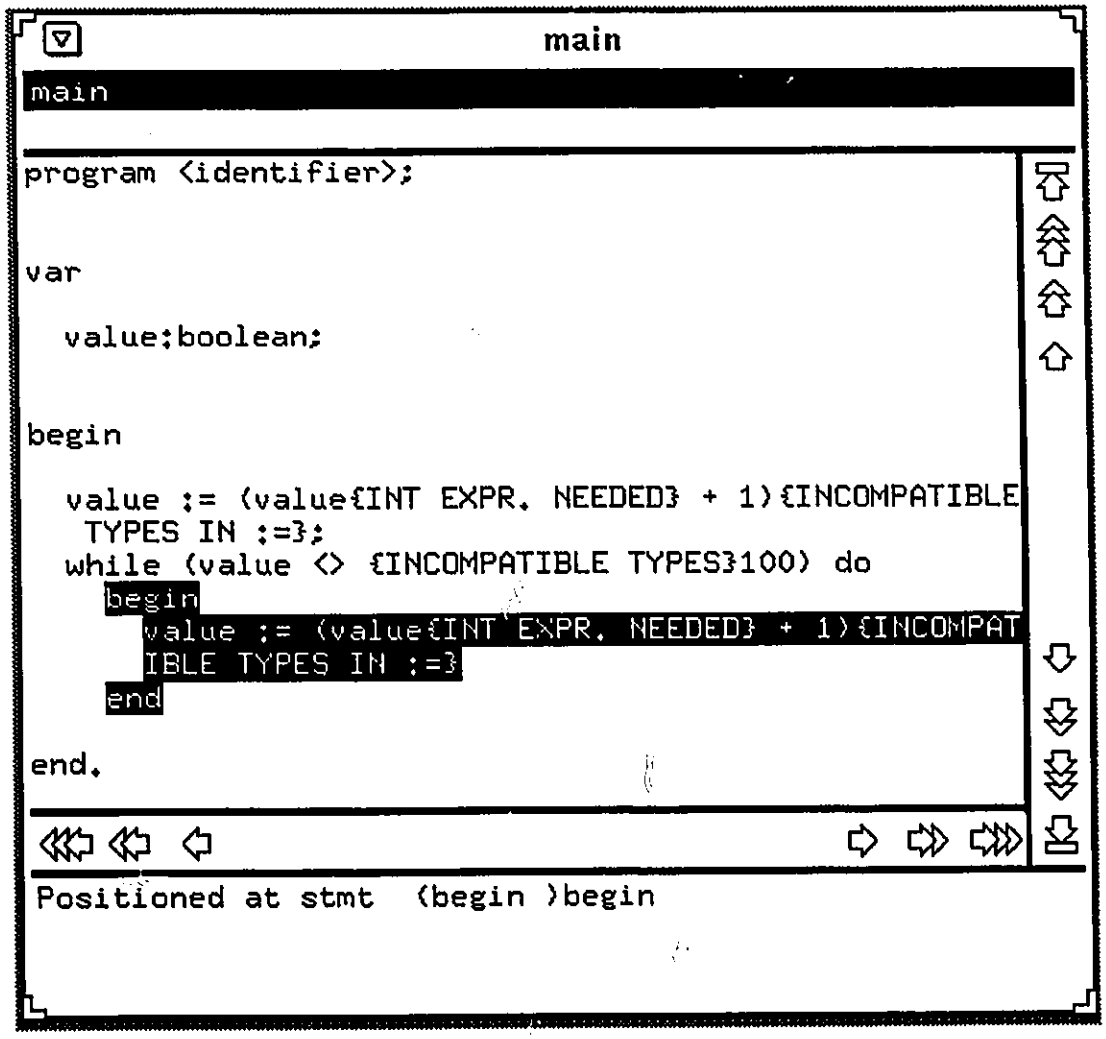

Figure 1.9

We can revert to Figure 1.10 by clicking on **)begin**. As we can see our program has some warning messages, we can remove them by declaring the variable **value** to have type **integer**. We can do this by first selecting **boolean**, then deleting it and then choosing **integer** from the menu which appears at the bottom of the screen. As soon as we perform the above actions all the warning messages disappear as seen in Figure 1.12.

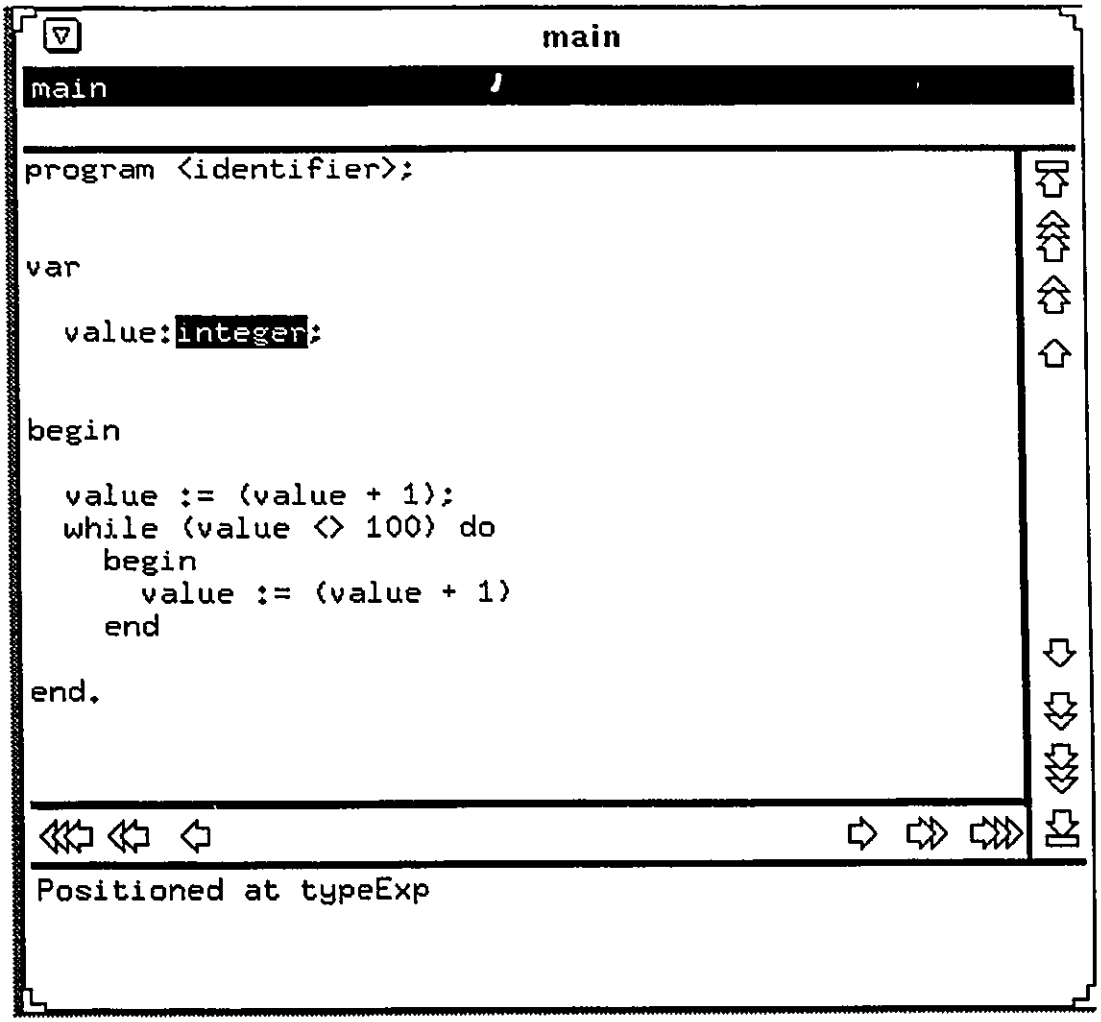

Figure 1.10

If we enter a name for the program at the remaining identifier placeholder and compile this program, we can be sure that our program will compile without coming up with any syntax or semantic error messages.

The above editing session with a structure editor illustrates some of the

features provided by a structure editor. The structure editor used in this example performs type checking, pretty-printing, transformations, checked for undeclared variables. It is possible to create sophisticated structure editors to perform incremental code generation, couple testing and debugging with editing *etc.*

If we have a powerful structure editor, one that performs syntax checking, semantic checking, type checking, incrementally generates code, and allows testing during editing, it is highly plausible that the time required for program development can be significantly reduced, because the programmer can focus on the intellectually challenging aspects of programming rather than waste time in debugging syntax and semantic errors.

APPENDIX 2
An Example Session with WAGE-ed

This section demonstrates the behavior of **WAGE-ed**. Specifically, it illustrates in detail how an attribute grammar specification is developed using **WAGE-ed**, the various warning messages issued by **WAGE-ed**, how to introduce regular and optional templates and placeholders and enter text.

The attribute grammar specification of Figure 3.2 will be implemented in W/AGE using **WAGE-ed**. The editor used to enter the specification is *unguarded* **wage_editor** and can be invoked on the School of Computer Science network by typing **xwageedit** at the system prompt.

When the editing session is initiated, the screen containing a template for the W/AGE program appears as seen in Figure 2.1.

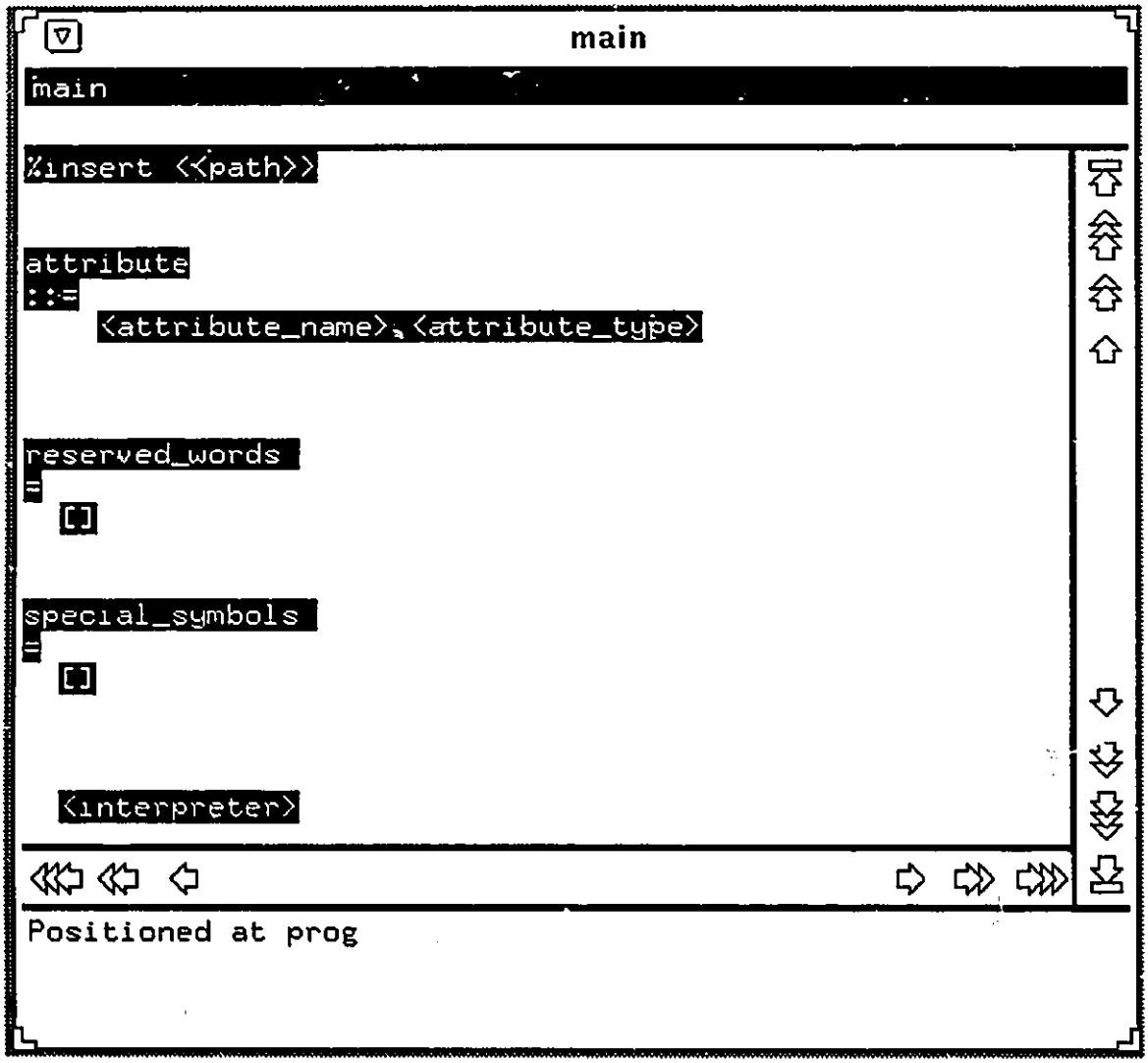

Figure 2.1

In Figure 2.1, the text which appears in bold font are W/AGE keywords and are immutable, italic text which appears in angle brackets are placeholders — locations which need to be refined. A placeholder should be first selected, in

order to refine it. It can be selected by placing the mouse arrow on it and clicking the left mouse button. As seen in Figure 2.1 above, initially the entire program template is selected.

The placeholders in **WAGE-ed** can be refined in an arbitrary order. In this example, initially the <*path*> placeholder is selected. Since the <*path*> placeholder has no transformations listed for it in the help pane, it can be refined by entering characters from the keyboard. In **WAGE-ed**, each component of the path name must be entered by introducing a new <*path*> placeholder, either by entering a sequence of returns or using the system command[5] ***forward-sibling-with-optionals***. To include the W/AGE header file, **local/header_for_WAGE_VERSION_2_RELEASE_0.m**, two <*path*> placeholders are required. Note that the / symbol is inserted automatically, if we enter the full path name it will not be accepted by the system. After entering the header file name, we select the special symbols declaration section to enter special symbols *viz.* :, [,], and , , see Figure 2.2 below.

---

5 For more explanation, refer to Appendix 3.

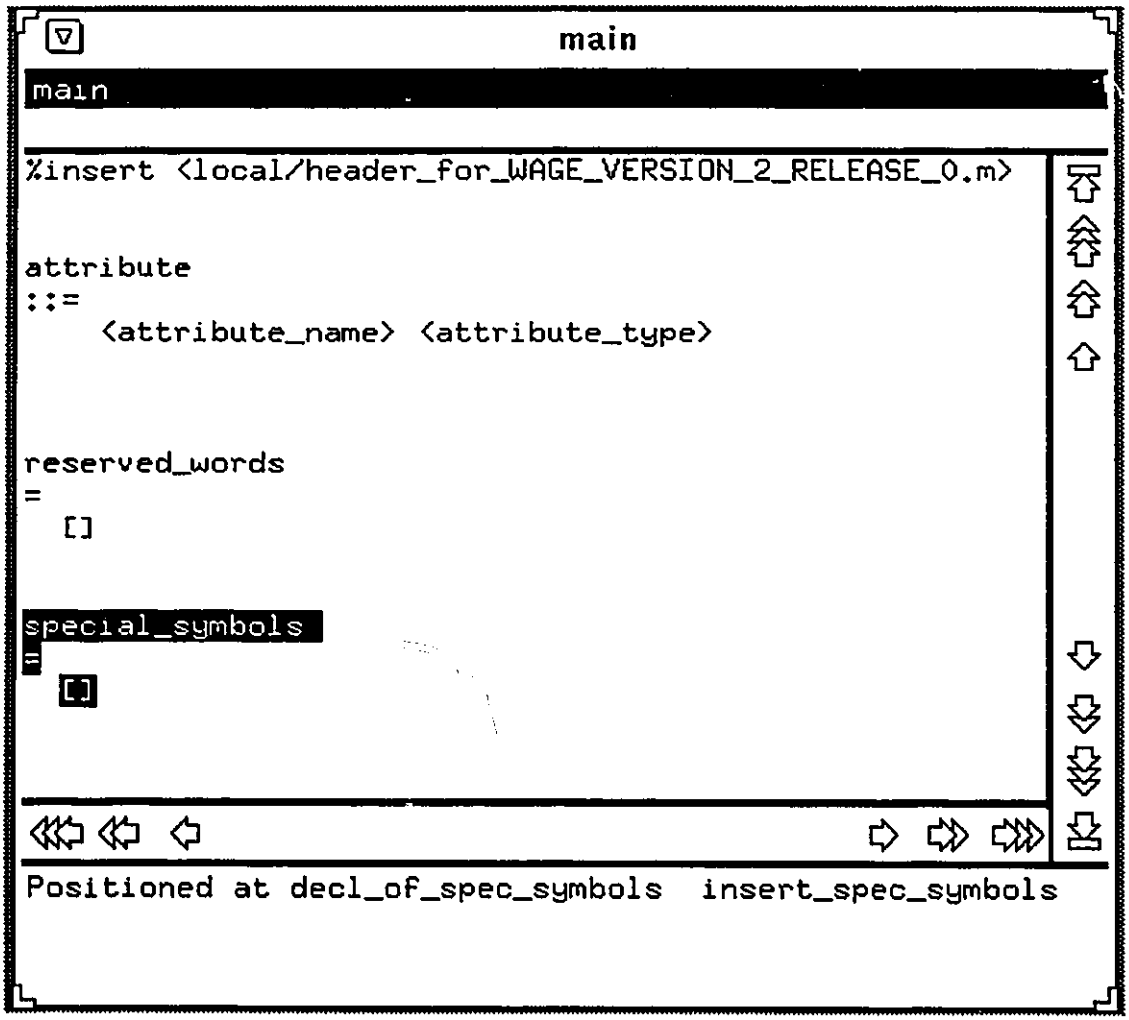

Figure 2.2

In order to transform the above selection to a template which contains a placeholder to enter special symbols, we click on **insert_spec_symbols** listed in the help pane. This introduces a *<spec_symbols>* placeholder between the opening and closing square bracket. A *<spec_symbols>* placeholder can be refined

by any single character, except alphanumeric characters. Placeholders for entering additional special symbols, after e .ering the first symbol, can be introduced by entering a sequence of return keys or using ***forward-sibling-with-optionals***. The result of entering :, [,], , and selecting <*interpreter*> for refinement is seen in Figure 2.3.

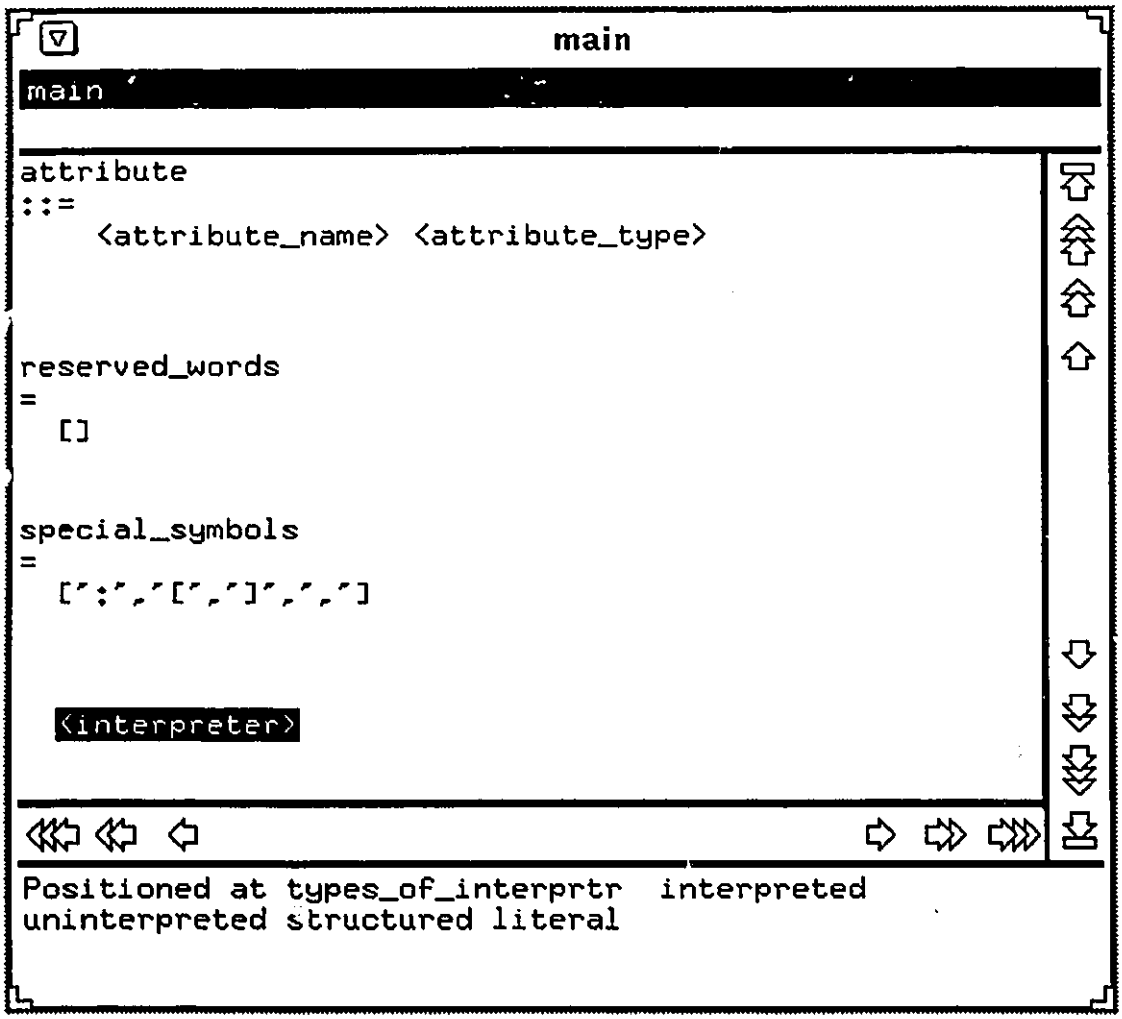

Figure 2.3

In Figure 2.3 above, the single quotes enclosing a symbol and the commas separating the symbol are inserted automatically by **WAGE-ed**. The *<interpreter>* placeholder has four transformations associated with it as seen listed in the help pane. The *<interpreter>* placeholder is refined into a template for structured in-

terpreter, by clicking on **structured** in the help pane, as seen in Figure 2.4 below.

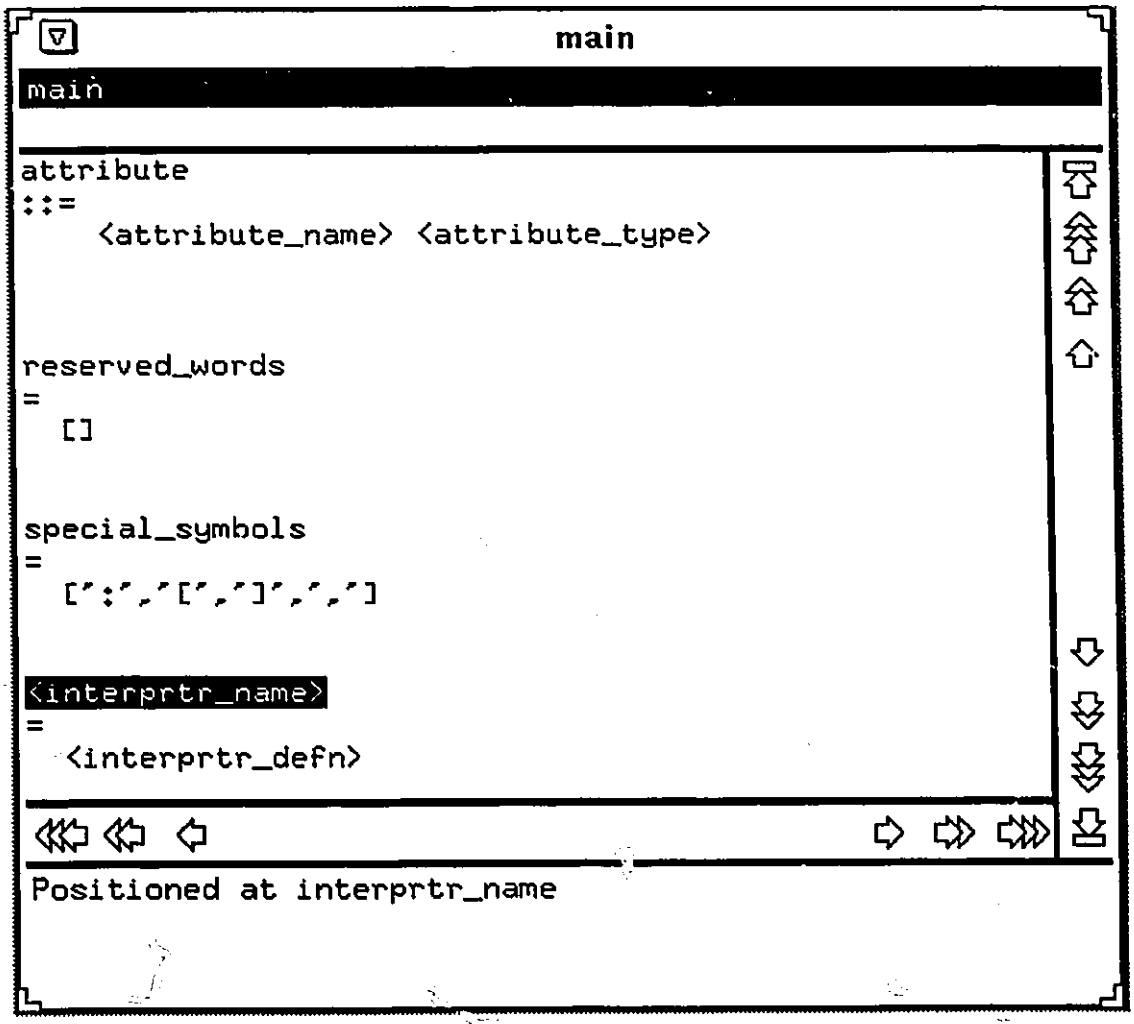

Figure 2.4

The template for entering structured interpreter definition consists of *<interprtr_name>* and *<interprtr_defn>* placeholders. As seen in the figure above the *<interprtr_defn>* placeholder has been automatically indented according to the

layout rule of Miranda. Since no transformations are associated with the <*interprtr_name*> placeholder, we can refine it by entering text from the keyboard. Placeholders for entering right hand side interpreter names and associated semantic rule can be introduced by selecting the <*interprtr_defn*> placeholder and then clicking on **show** listed in the help pane. The result of refining the <*interprtr_name*> placeholder and transforming the <*interprtr_defn*> placeholder is seen in Figure 2.5.

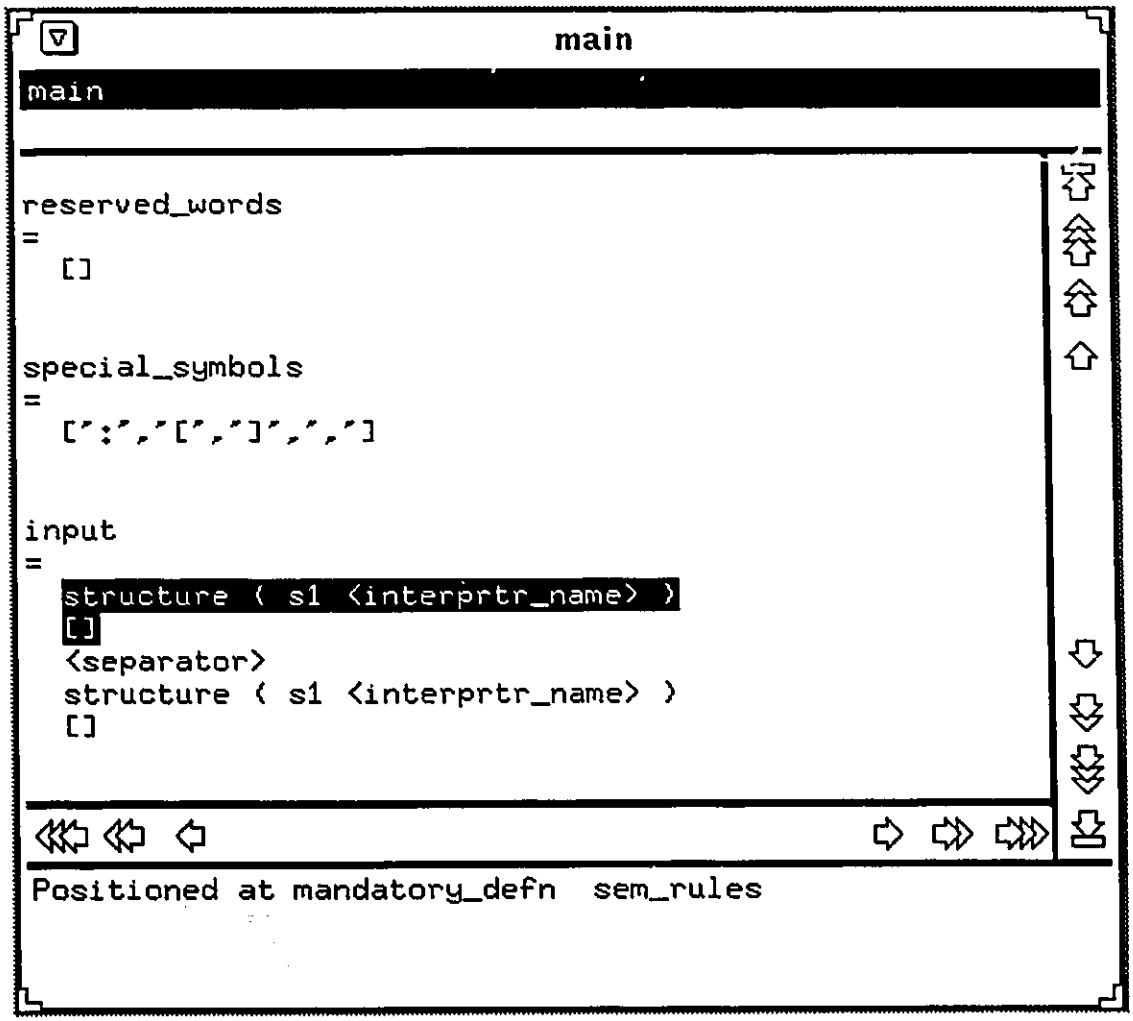

Figure 2.5

As seen in the figure above, the template for entering right hand side interpreter definition consists of two templates for entering right hand side production definition separated by the placeholder <*separator*>. Each template consists of <*interprtr_name*> placeholder, additional placeholders can be obtained by first

refining the already selected *<interprtr_name>* placeholder and then entering a sequence of return keys or using ***forward-sibling-with-optionals***. In Figure 2.5 above, we refine the first right hand side structure definition option by entering the interpreter names: **number, colon, op_bracket, list_of_numbers,** and finally **cl_bracket** and then select the <separator> placeholder. The result of the previous operation is seen in Figure 2.6.

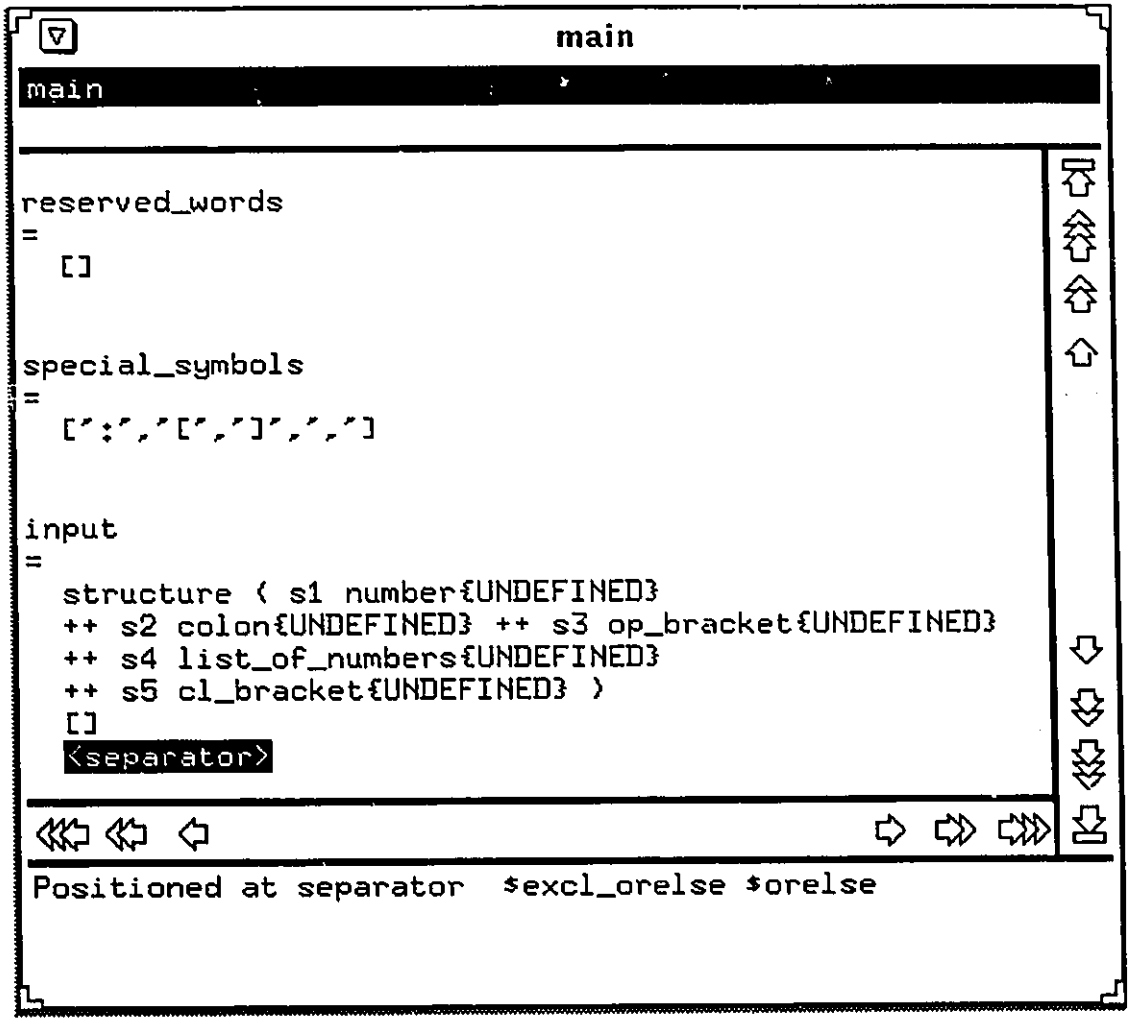

Figure 2.6

WAGE-ed requires that all right hand side interpreter names be declared before their use. In our case, since we have not declared the right hand side interpreter names, the interpreter names are juxtaposed with the message {UN-**DEFINED}**. **WAGE-ed** does not constraint the user to take care of the warning

messages immediately. Now, we refine the *<separator>* placeholder by choosing **$excl_orelse** from the help pane. Next, we refine the second structure template by entering the interpreter names: **string, colon, op_bracket, list_of_strings,** and finally **cl_bracket.** The result of previous operations is seen in figure below.

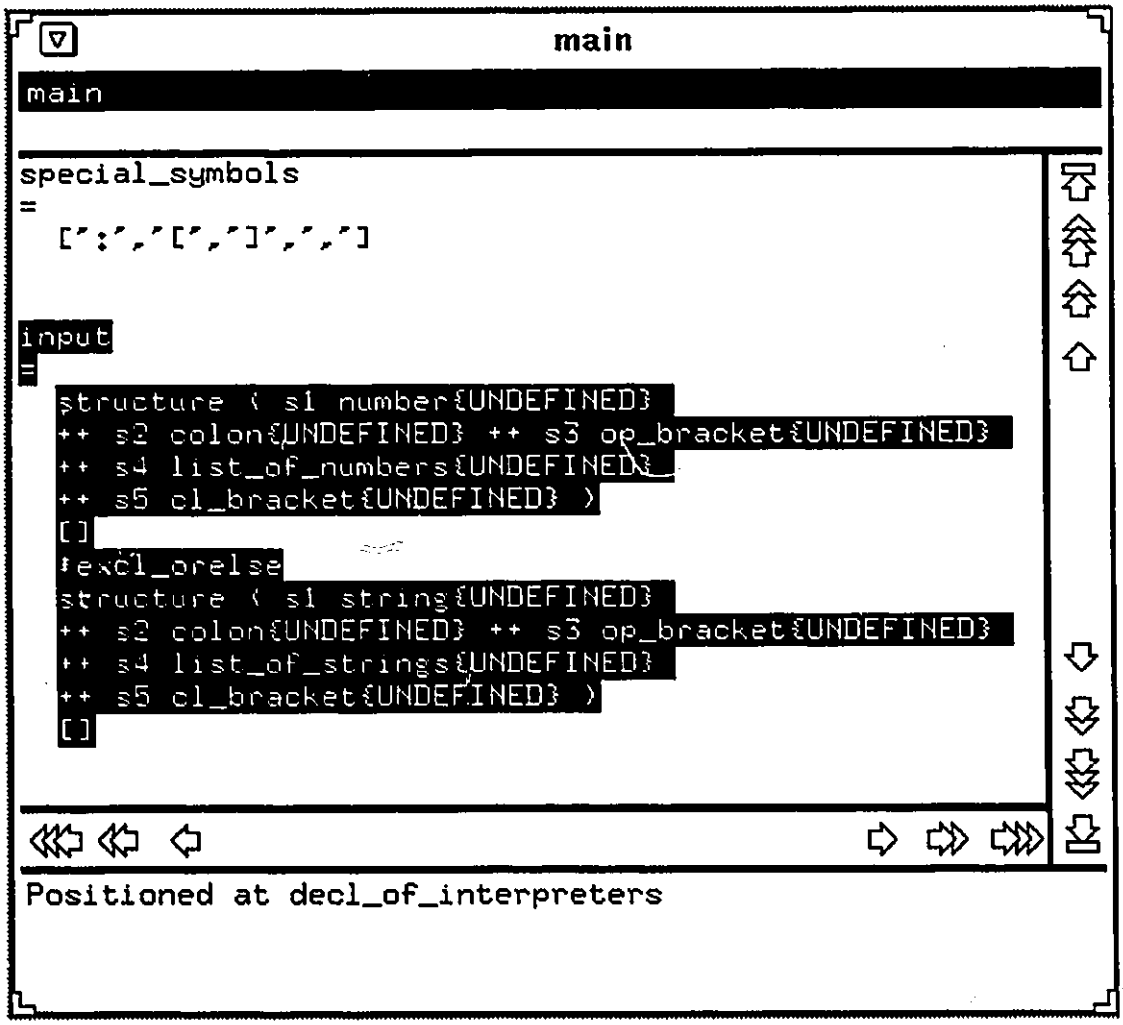

Figure 2.7

Next we declare all undefined interpreters. To declare an interpreter we need an *<interpreter>* placeholder to begin with. This can be done by selecting the entire definition of the interpreter **input**, by clicking on the equal sign, and then using the system command ***backward-sibling-with-optionals***, which will introduce an *<interpreter>* placeholder before the definition of the interpreter **input**. If we use ***forward-sibling-with-optionals***, an *<interpreter>* placeholder will be introduced after the definition of the interpreter **input**. Since we need to define interpreters before their use, we use ***backward-sibling-with-optionals***.

The interpreters **number**, **list_of_numbers**, **string**, **list_of_strings**, **colon**, **op_bracket**, and **cl_bracket** have been defined as **literal**, **structured**, **literal**, **structured**, **uninterpreted**, **uninterpreted**, and **uninterpreted** interpreters respectively, as shown in Figure 2.8.

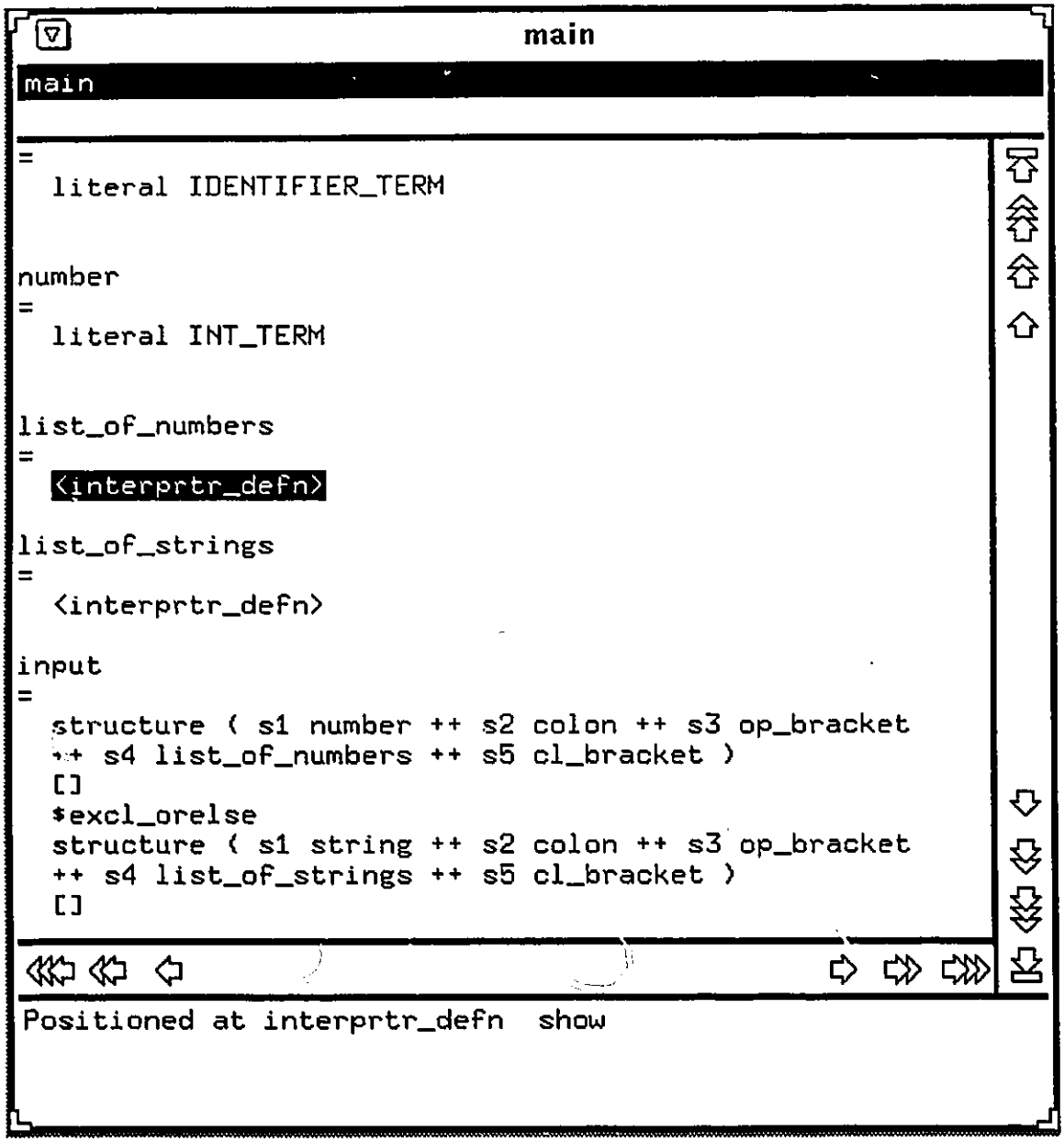

Figure 2.8

Note in the figure above all the undefined messages have disappeared. Next

we refine the two <*interprtr_defn*> placeholders in the figure above and define the **comma** as an **uninterpreted** interpreter. The resulting structure is seen in Figure 2.9.

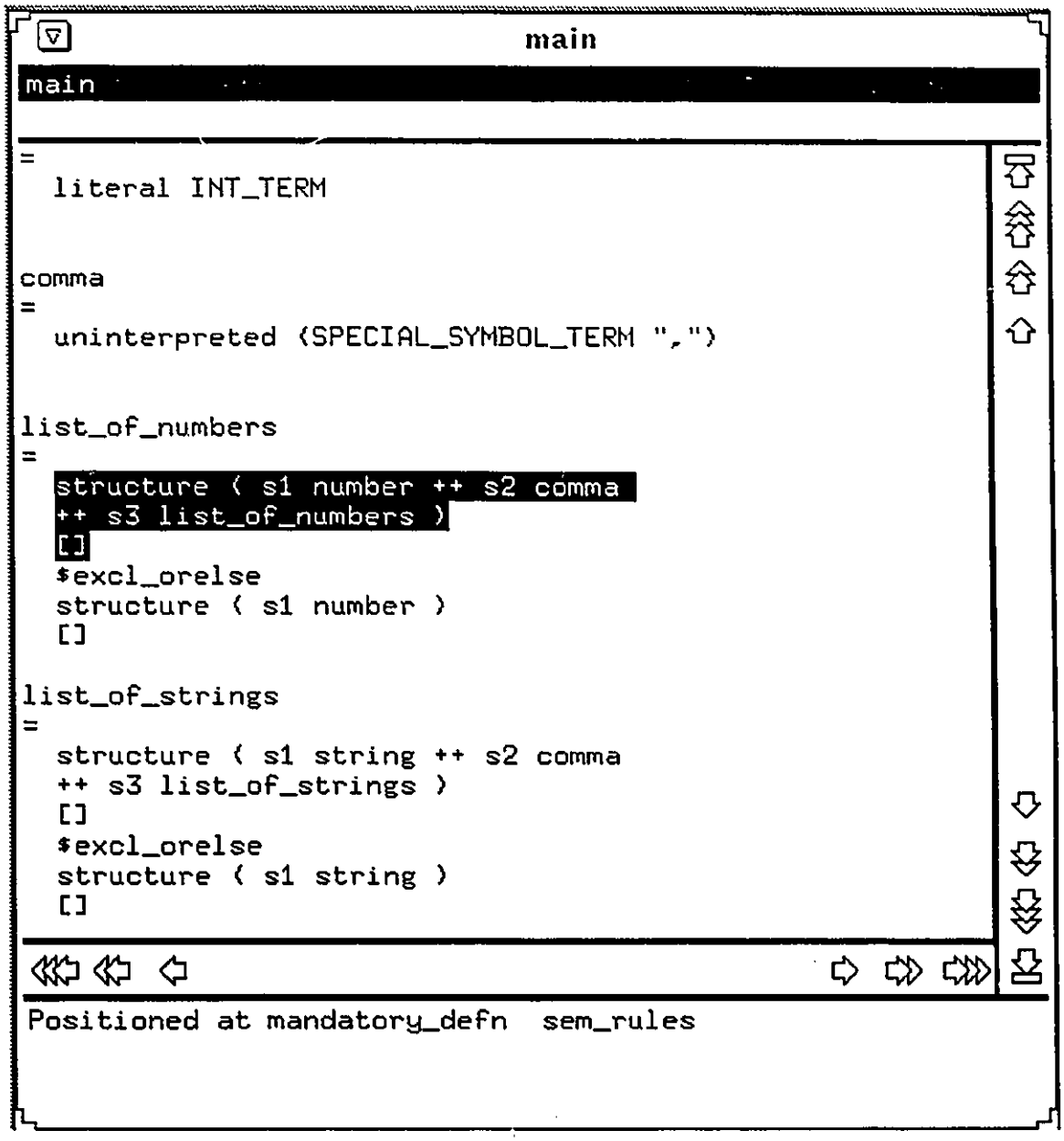

Figure 2.9

Up to this stage, our program is syntactically correct. The readers must

have noticed, at each stage the program was well-formed, **WAGE-ed** prohibited any syntax errors, insertion of all punctuation symbols, keywords and layout was done by **WAGE-ed** as a result very minimal typing is required on part of the programmer and at the same time syntactical integrity of the program was maintained.

Semantic rules associated with a production can be inserted by first selecting the production, this is done by clicking on the keyword **structure**. In the previous figure we selected the first right hand side production of the interpreter **list_of_numbers**. Placeholder for entering a semantic rule, <*semantic_rule*>, is inserted by clicking on **sem_rules** displayed in the help pane. After following a sequence of transformation, <*semantic_rule*> can be transformed into a template for defining an attribute equation for synthesized or inherited attribute. Templates for entering additional attribute equations can be obtained in a similar fashion by first introducing the <*semantic_rule*> placeholder, this can be done by first selecting a previously entered attribute equation and then entering a sequence of return keys or using the system commands ***forward-sibling-with-optionals*** or ***backward-sibling-with-optionals***. Figure 2.10 shows the result of entering the semantic rules associated with the first production for the interpreter **list_of_numbers**.

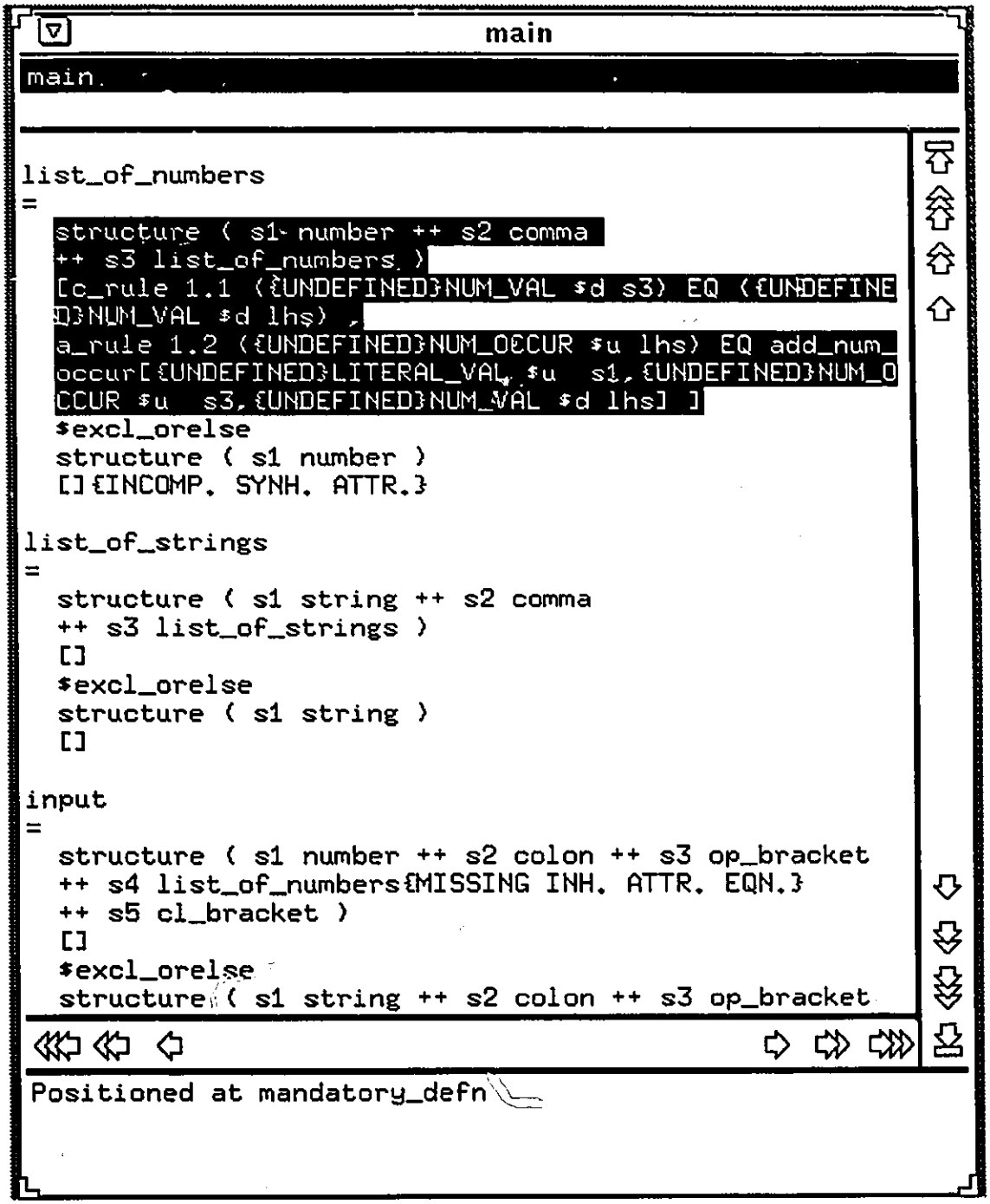
main
main.
list_of_numbers
=
structure (s1 number ++ s2 comma
++ s3 list_of_numbers)
[c_rule 1.1 ({UNDEFINED}NUM_VAL $d s3) EQ ({UNDEFINED}NUM_VAL $d lhs) ,
a_rule 1.2 ({UNDEFINED}NUM_OCCUR $u lhs) EQ add_num_occur[{UNDEFINED}LITERAL_VAL $u s1,{UNDEFINED}NUM_OCCUR $u s3,{UNDEFINED}NUM_VAL $d lhs]]
$excl_orelse
structure (s1 number)
[]{INCOMP. SYNH. ATTR.}
list_of_strings
=
structure (s1 string ++ s2 comma
++ s3 list_of_strings)
[]
$excl_orelse
structure (s1 string)
[]
input
=
structure (s1 number ++ s2 colon ++ s3 op_bracket
++ s4 list_of_numbers{MISSING INH. ATTR. EQN.}
++ s5 cl_bracket)
[]
$excl_orelse
structure (s1 string ++ s2 colon ++ s3 op_bracket
Positioned at mandatory_defn

Figure 2.10

As seen in the figure above, the program has been annotated with various warning messages. The attributes names are juxtaposed with the message **{UNDEFINED}** because we have not defined them as yet in the attribute declaration section, the second production for interpreter **list_of_numbers** is annotated with the message **{INCOMP. SYNH. ATTR.}** because the production does not have a semantic rule for the synthesized attribute **NUM_OCCUR**, and finally the message **{MISSING INH. ATTR. EQN.}** appears because we have not defined an attribute equation for the inherited attribute **NUM_VAL** to be passed down the interpreter **list_of_numbers** *i e.* **s4** in definition for interpreter **input** whereas it is used in semantic rule 1.1.

In order to remove these messages, we:

- Declare the attributes: **LITERAL_VAL, NUM_VAL, NUM_OCCUR,** and **STR_VAL** in the attribute declaration section,
- Define an attribute equation for the synthesized attribute **NUM_OCCUR,** and
- Define an attribute equation for the inherited attribute **NUM_VAL.**

The result of performing the previous operation and entering semantic rules for interpreter **list_of_strings** is seen in Figure 2.11.

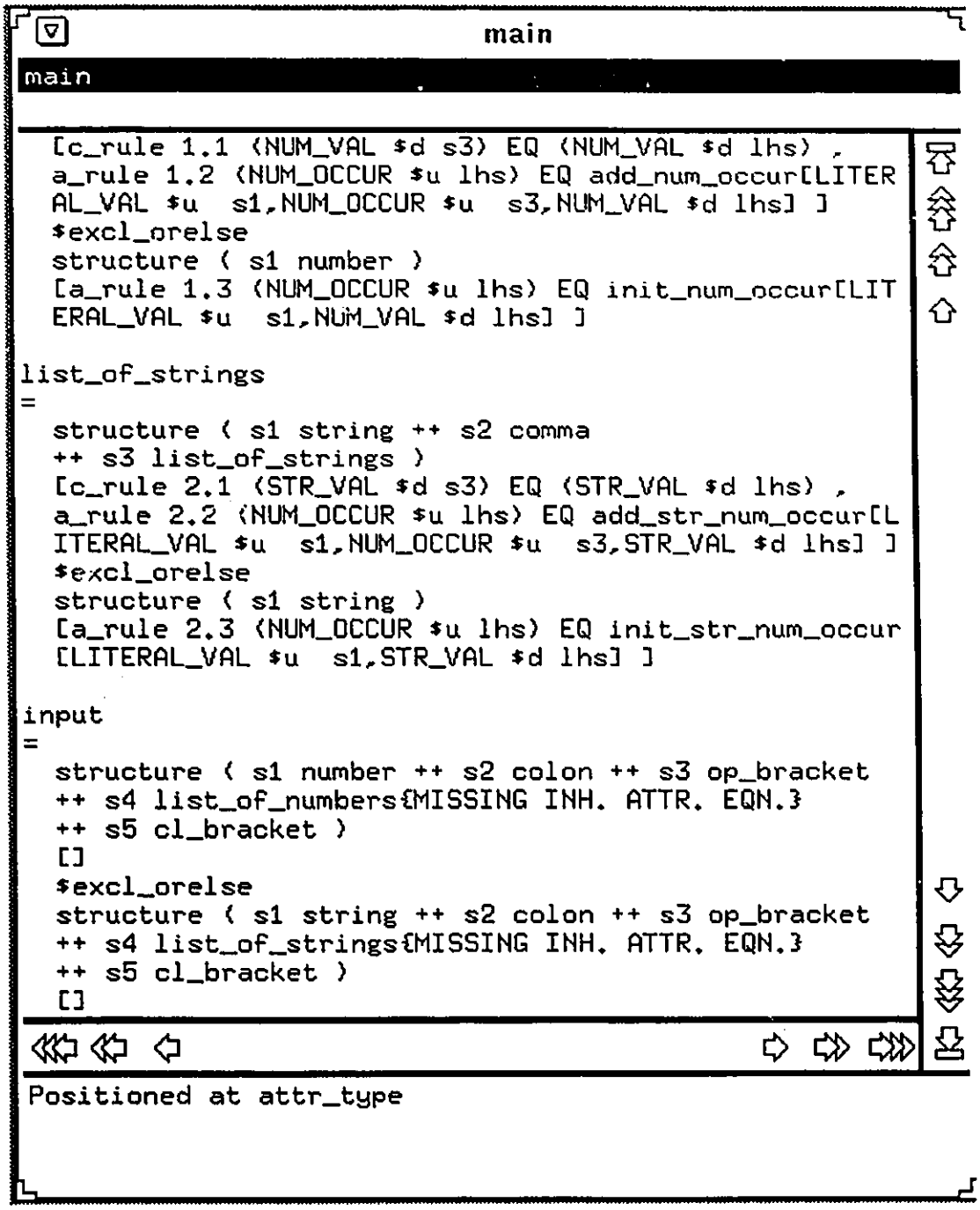
main
main
[c_rule 1.1 (NUM_VAL $d s3) EQ (NUM_VAL $d lhs) ,
a_rule 1.2 (NUM_OCCUR $u lhs) EQ add_num_occur[LITER
AL_VAL $u s1,NUM_OCCUR $u s3,NUM_VAL $d lhs]]
$excl_orelse
structure (s1 number)
[a_rule 1.3 (NUM_OCCUR $u lhs) EQ init_num_occur[LIT
ERAL_VAL $u s1,NUM_VAL $d lhs]]
list_of_strings
=
structure (s1 string ++ s2 comma
++ s3 list_of_strings)
[c_rule 2.1 (STR_VAL $d s3) EQ (STR_VAL $d lhs) ,
a_rule 2.2 (NUM_OCCUR $u lhs) EQ add_str_num_occur[L
ITERAL_VAL $u s1,NUM_OCCUR $u s3,STR_VAL $d lhs]]
$excl_orelse
structure (s1 string)
[a_rule 2.3 (NUM_OCCUR $u lhs) EQ init_str_num_occur
[LITERAL_VAL $u s1,STR_VAL $d lhs]]
input
=
structure (s1 number ++ s2 colon ++ s3 op_bracket
++ s4 list_of_numbers{MISSING INH. ATTR. EQN.3
++ s5 cl_bracket)
[]
$excl_orelse
structure (s1 string ++ s2 colon ++ s3 op_bracket
++ s4 list_of_strings{MISSING INH. ATTR. EQN.3
++ s5 cl_bracket)
[]
Positioned at attr_type

Figure 2.11

As seen in the figure above, all the warning messages have disappeared, except for one. Next we introduce placeholders for entering semantic rules for each right hand side production of the interpreter **input** by clicking on the keyword **structure** associated with each production. Due to limitation of space, the entire program is shown in Figures 2.12, 2.13, and 2.14.

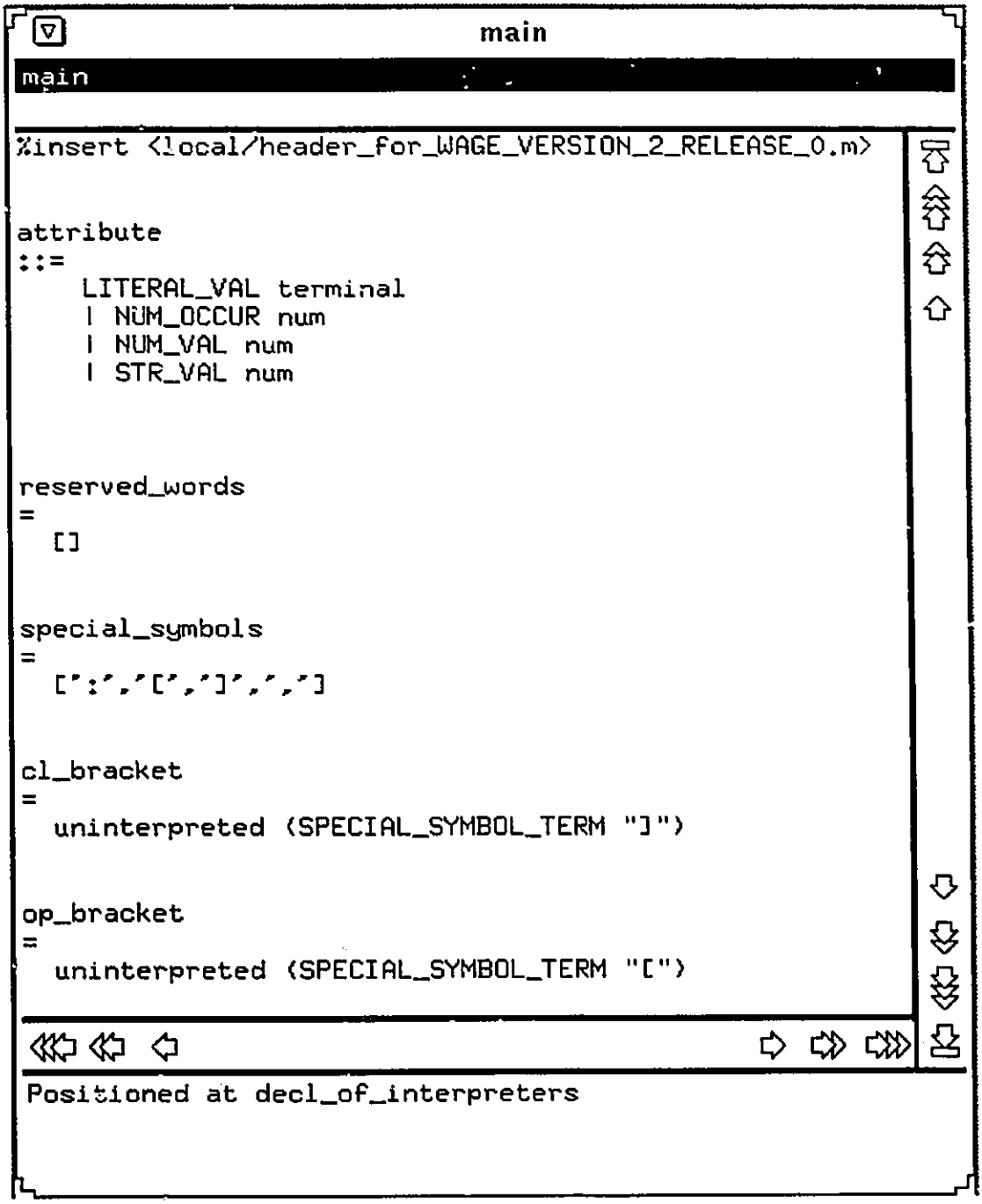
main
main
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
attribute
::=
LITERAL_VAL terminal
| NUM_OCCUR num
| NUM_VAL num
| STR_VAL num
reserved_words
=
[]
special_symbols
=
[":","[","]",",","]
cl_bracket
=
uninterpreted (SPECIAL_SYMBOL_TERM "]")
op_bracket
=
uninterpreted (SPECIAL_SYMBOL_TERM "[")
Positioned at decl_of_interpreters

Figure 2.12

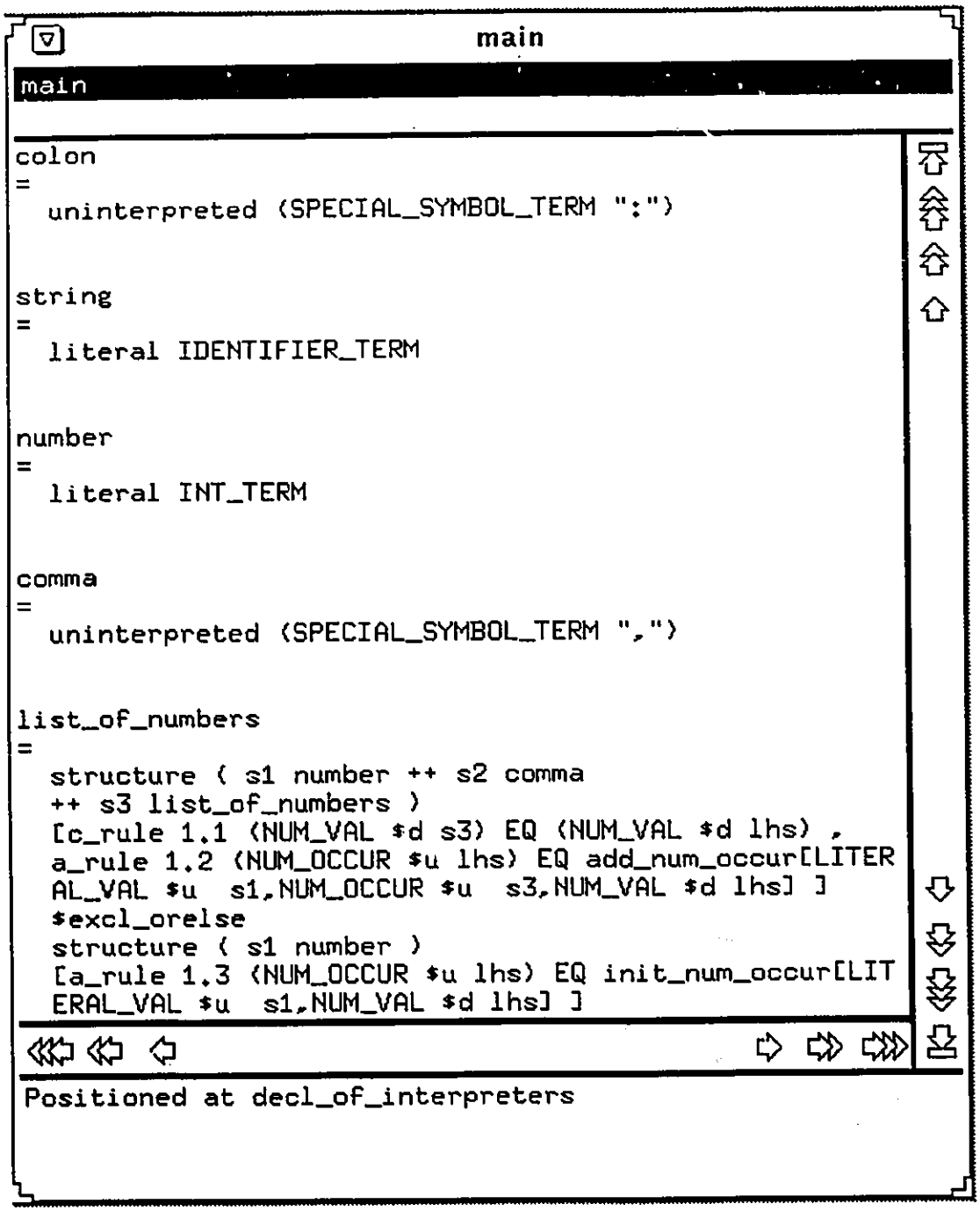
main
main
colon
=
uninterpreted (SPECIAL_SYMBOL_TERM ":")
string
=
literal IDENTIFIER_TERM
number
=
literal INT_TERM
comma
=
uninterpreted (SPECIAL_SYMBOL_TERM ",")
list_of_numbers
=
structure (s1 number ++ s2 comma
++ s3 list_of_numbers)
[c_rule 1.1 (NUM_VAL $d s3) EQ (NUM_VAL $d lhs) ,
a_rule 1.2 (NUM_OCCUR $u lhs) EQ add_num_occur[LITER
AL_VAL $u s1,NUM_OCCUR $u s3,NUM_VAL $d lhs]]
$excl_orelse
structure (s1 number)
[a_rule 1.3 (NUM_OCCUR $u lhs) EQ init_num_occur[LIT
ERAL_VAL $u s1,NUM_VAL $d lhs]]
Positioned at decl_of_interpreters

Figure 2.13

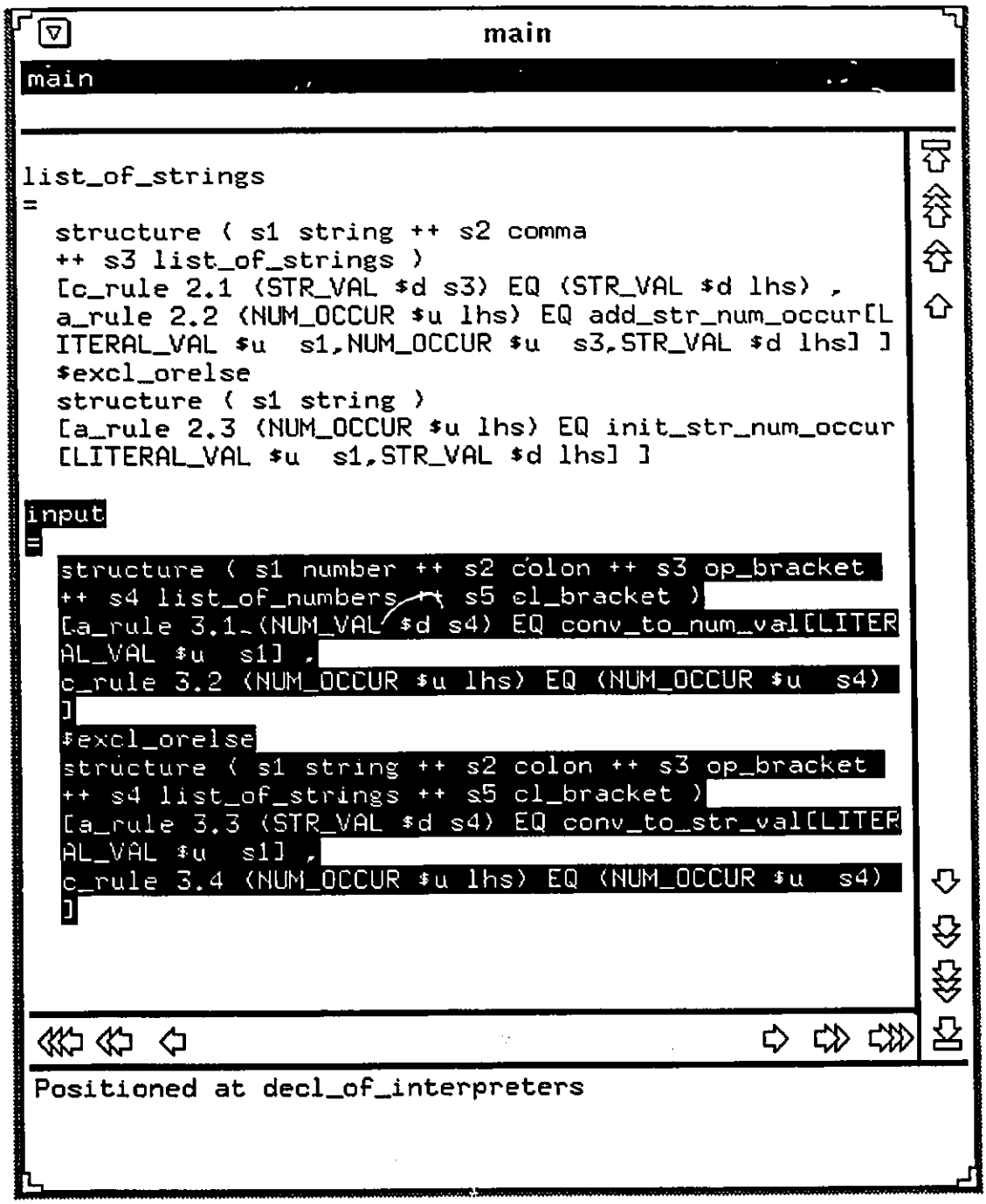
main
main
list_of_strings
=
structure (s1 string ++ s2 comma
++ s3 list_of_strings)
[c_rule 2.1 (STR_VAL $d s3) EQ (STR_VAL $d lhs) ,
a_rule 2.2 (NUM_OCCUR $u lhs) EQ add_str_num_occur[L
ITERAL_VAL $u s1,NUM_OCCUR $u s3,STR_VAL $d lhs]]
$excl_orelse
structure (s1 string)
[a_rule 2.3 (NUM_OCCUR $u lhs) EQ init_str_num_occur
[LITERAL_VAL $u s1,STR_VAL $d lhs]]
input
=
structure (s1 number ++ s2 colon ++ s3 op_bracket
++ s4 list_of_numbers ++ s5 cl_bracket)
[a_rule 3.1 (NUM_VAL $d s4) EQ conv_to_num_val[LITER
AL_VAL $u s1] ,
c_rule 3.2 (NUM_OCCUR $u lhs) EQ (NUM_OCCUR $u s4)
]
$excl_orelse
structure (s1 string ++ s2 colon ++ s3 op_bracket
++ s4 list_of_strings ++ s5 cl_bracket)
[a_rule 3.3 (STR_VAL $d s4) EQ conv_to_str_val[LITER
AL_VAL $u s1] ,
c_rule 3.4 (NUM_OCCUR $u lhs) EQ (NUM_OCCUR $u s4)
]
Positioned at decl_of_interpreters

Figure 2.14

If this program is compiled, the only error messages signalled will be the undefined function names: **add_str_num_occur**, **init_str_num_occur**, **add_num_occur**, **init_num_occur**, **conv_to_num_val**, and **conv_to_str_val**, because **WAGE-ed** does not support any editing facility for entering of Miranda functions.

§ 3.1 Notations Used

The notations used in this appendix are Miranda notations whose meanings are explained below:

- x == y, introduces x as an acronym for the type name y,
- x :: y, declares x to be of type y,

where **type** is defined inductively as follows:

num, char, bool ϵ **type.**

If **t** ϵ **type,** then so is **[t]** *i.e.* list type whose elements are of type **t.**

If $t_1,..., t_n \epsilon$ **type,** then so is $(t_1,..., t_n)$ *i.e.* tuple type with elements of type t_1 to t_n.

if $t_1, t_2 \epsilon$ **type,** then so is $t_1 \longrightarrow t_2$ *i.e.* a function that accepts input argument of type t_1 and output result of type t_2.

If **y, z** ϵ **type** then so is x, where x ::= C_1 y | ...| C_n z, and C_1 to C_n are user-defined *constructors.*

§ 3.2 Components of W/AGE

W/AGE consists of the following components:

- A function for applying interpreters: ***apply_interpreter***,

- A lexical scanning function: ***tokenise***,
- A set of functions for building basic interpreters: ***literal***, ***interpreted***, and ***uninterpreted***,
- A set of interpreter combinators: ***$orelse***, ***$excl_orelse***, ***$enables***, and ***structure***.

3.2.1 Type of Interpreters

In W/AGE the type ***interpreter*** is defined as:

interpreter == [([attribute], [terminal], [terminal])] —> [([attribute], [terminal], [terminal])]

The above definition means that an interpreter is a function that maps a list of triples of type [([attribute], [terminal], [terminal])] to a list of triples of the same type, such that:

- Each triple (as, ts_1, ts_2) in the list that is input to an interpreter is such that the list of attributes as may be regarded as a context in which the list of terminals ts_2 is to be interpreted,
- Each triple (as', ts^1, ts^2) in the list that is output by an interpreter is associated to exactly on pair (as, ts_1, ts_2) in the input list such that: (i) as' is a subset of the union of as and some initial segment of ts_2, (ii) ts^2 is the list of remaining uninterpreted terminals in ts_2, and (iii) ts^1 is the list of terminals that have been consumed,

• Interpreters return lists of triples because each pair in the input may have more than one interpretation.

3.2.2 Type of Terminals

W/AGE classifies terminals into the following categories:

• **INT_TERM**: includes all integer numbers.

• **REAL_TERM**: includes all real numbers.

• **IDENTIFIER_TERM**: includes all legal identifiers — alphanumeric characters, starting with an alphabet, may have embedded underscores and no blanks.

• **SPECIAL_SYMBOL_TERM**: includes all single character symbols, except alphabet and digits.

• **RESERVED_WORD_TERM**: includes strings which are declared in the reserved declaration section.

• **ANY_TERM**: could be any of the above mentioned types.

• **UNCATEGORISED_TERM**: any term which does not fall into any of the categories mentioned above.

3.2.2.1 terminal

The type of ***terminal*** is defined in W/AGE as:

```
terminal ::= INT_TERM [char]
           | REAL_TERM [char]
           | IDENTIFIER_TERM [char]
           | SPECIAL_SYMBOL_TERM [char]
           | RESERVED_WORD_TERM [char]
           | ANY_TERM [char]
           | UNCATEGORISED_TERM [char]
```

3.2.3 Type of Functions for Top-level Application of Interpreters

The ***apply_interpreter*** function in W/AGE is defined as follows:

```
apply_interpreter ::interpreter -> string_to_be_interpreted
                  -> [([attribute], [terminal])]
```

The above definition means that the ***apply_interpreter*** function takes as input an ***interpreter*** (of type mentioned above) and the string to be interpreted (of type [char]) and returns a list of pairs of type ([attribute], [terminals]). According to the symbols used in the definition of ***interpreter***, ([attribute], [terminals]) would be (as', ts^2) respectively.

Usage

apply_interpreter **interpreter_name** " string_to_be_interpreted "

Example

apply_interpreter **input** "1:[1,2,1,3,1]"

3.2.4 Type of Lexical Scanning Function

W/AGE provides a function, ***tokenise***, for classifying terminals according to their type.

Type

```
tokenise :: string_to_be_processed -> [terminal]
            where string_to_be_processed == [char]
```

Usage

tokenise "string_to_be_processed"

Example

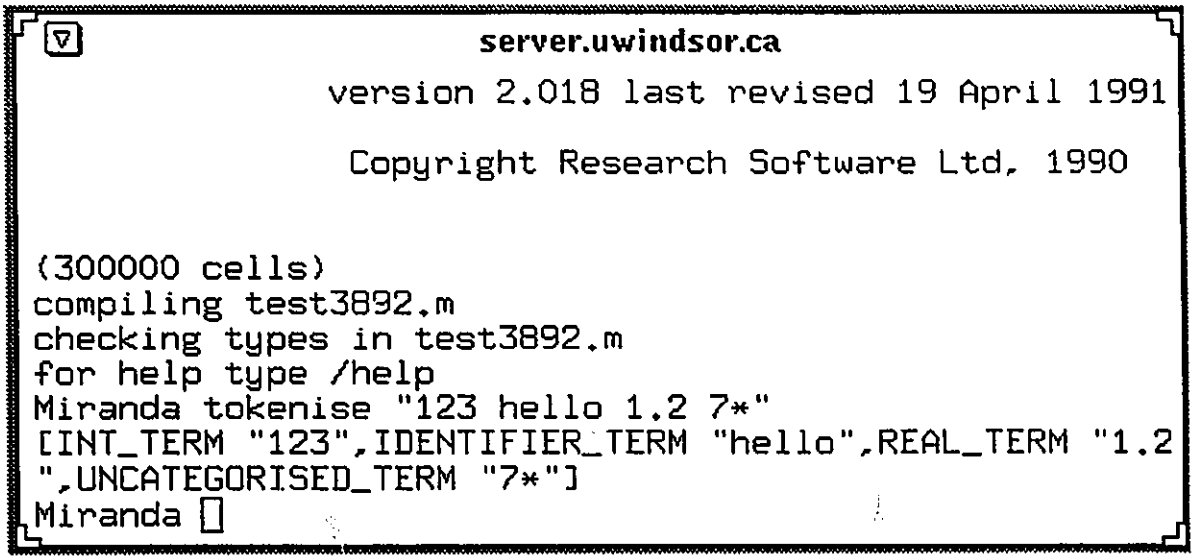

Figure 3.1

3.2.5 Type of Functions for Building Basic Interpreters

In W/AGE there are three functions that may be used to build interpreters for single terminals, namely,

- ***literal***

- *uninterpreted*
- *interpreted*

3.2.5.1 literal

The interpreter constructor ***literal*** accepts a string and a terminal and returns an interpreter. If an interpreter is defined as a ***literal*** interpreter then it has automatically associated with it **only one** synthesized attribute **LITERAL_VAL** of type **terminal**, which is as defined in section 4.3.2. It cannot have any inherited attributes, nor can it have any other synthesized attributes.

Type

```
[char] -> terminal -> interpreter
```

Usage

interpreter_name = ***literal*** *term_declaration*

where *term_declaration*, could be any one of:

- **INT_TERM** : matches any integer number,
- **REAL_TERM** : matches any real number,
- **RESERVED_WORD_TERM** : matches any word which has been declared as a reserved word,
- **SPECIAL_SYMBOL_TERM** : matches any single character symbols, except alphabet and digit, and which have been declared as special symbols,

• **IDENTIFIER_TERM** : matches any identifier (can be composed of any alphanumeric characters with embedded underscores and no blanks),

• **ANY_TERM** : matches any thing which belongs to any of the categories mentioned above,

• **UNCATEGORISED_TERM**: matches any thing which does not belong to any of the term types mentioned above.

Example

```
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
attribute
::=
   LITERAL_VAL terminal
special_symbols
=
 ['+','-']
reserved_words
=
 ["begin", "end"]
type_of_terminals
=
 literal SPECIAL_SYMBOL_TERM
 $excl_orelse
 literal RESERVED_WORD_TERM
 $excl_orelse
 literal IDENTIFIER_TERM
 $excl_orelse
 literal INT_TERM
 $excl_oresle
 literal REAL_TERM
```

Figure 3.2

After compiling the above program, if we apply the function ***apply_interpreter*** to the interpreter **type_of_terminals** with an argument, then the synthesized

attribute **LITERAL_VAL** associated with **type_of_terminals** (it is defined as a ***literal*** interpreter) will contain the value of the argument. See Figure 4.3.

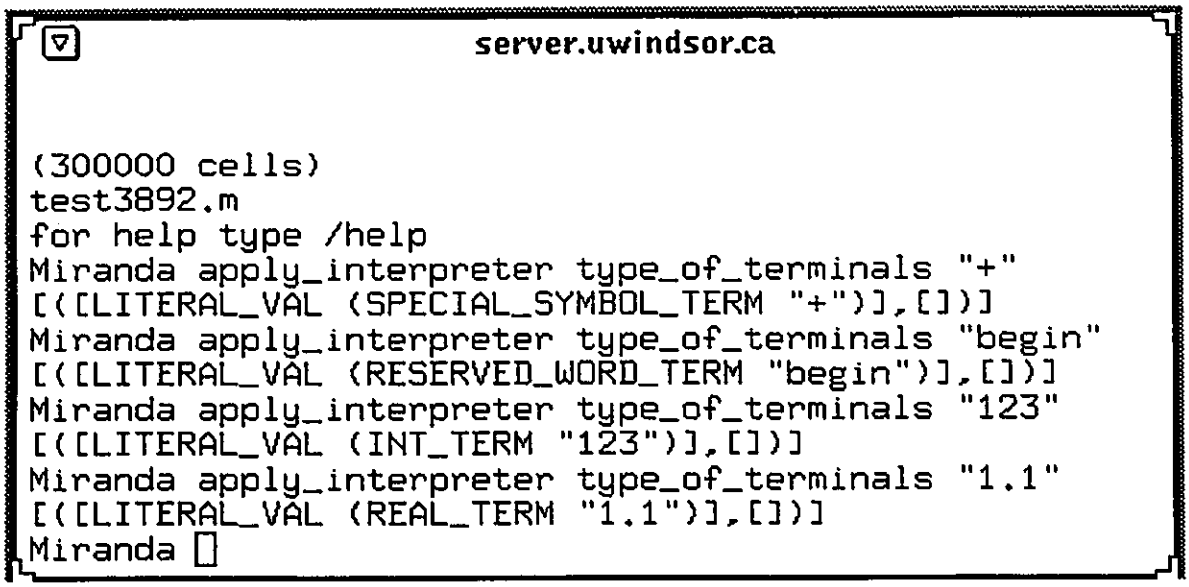

Figure 3.3

3.2.5.2 uninterpreted

The interpreter constructor ***uninterpreted*** accepts a terminal and returns an interpreter. If an interpreter is defined as an ***uninterpreted*** interpreter, then it has no attributes associated with it. An interpreter is defined as ***uninterpreted*** if we only want to match a terminal (string) in the input but do not need its value in processing the input. Terminals such as keywords (begin, end, while *etc.*), punctuation symbols are usually defined as **uninterpreted.**

Type

```
terminal -> interpreter
```

Usage

interpreter_name = ***uninterpreted*** (*term_declaration*)

where *term_declaration* could be any one of the following:

• **RESERVED_WORD_TERM** " *reserved_word* ": *reserved_word* is declared in Reserved word declaration section,

• **SPECIAL_SYMBOL_TERM** " *special_symbol* ": *special_symbol* is declared in Special symbol declaration section,

• **INT_TERM** " *any_integer* "

• **INT_TERM any**

• **REAL_TERM** " *any_real* "

• **REAL_TERM any**

• **IDENTIFIER_TERM** " *any_identifier* "

• **IDENTIFIER_TERM any**

• **ANY_TERM** " *any_terminal* "

• **ANY_TERM any**

The keyword **any** acts as a wild-card, meaning that it will match any terminal in the input if it is of type similar to the constructor with which the corresponding **any** is associated.

Example

```
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
attribute
::=
   LITERAL_VAL terminal
special_symbols
=
  ['+','-']
reserved_words
=
  ["begin", "end"]
type_of_terminals
=
  uninterpreted SPECIAL_SYMBOL_TERM "+"
  $excl_orelse
  uninterpreted RESERVED_WORD_TERM "begin"
  $excl_orelse
  uninterpreted IDENTIFIER_TERM any
  $excl_orelse
  uninterpreted INT_TERM "123"
  $excl_oresle
  uninterpreted REAL_TERM any
```

Figure 3.4

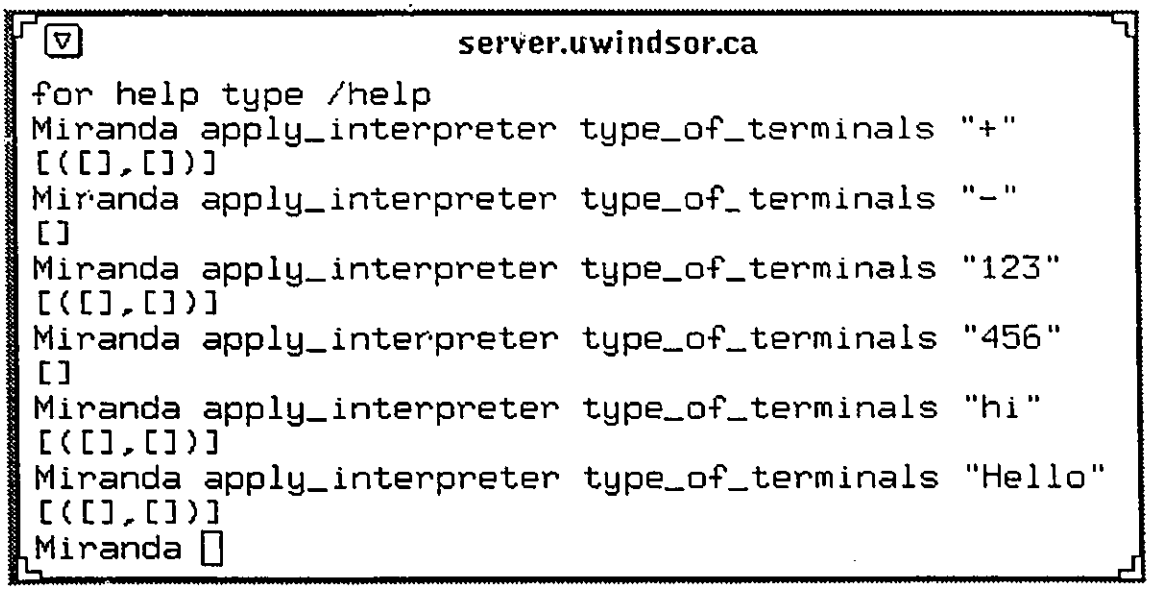

Figure 3.5

In Figure 4.5 above,

- [([],[])] denotes that the terminal (string) has been accepted,
- [] denotes that the terminal (string) was not accepted.

The terminal **456** was not accepted, because we have defined our interpreter **type_of_terminals** to match only that integer whose value is **123**. In the other case, the interpreter successfully accepted identifiers **hi** and **Hello**, because we have defined it to match *any* legal identifier.

3.2.5.3 interpreted

The interpreter constructor ***interpreted*** accepts the pair (terminal, [attribute]) and returns an interpreter. If an interpreter is defined as an ***interpreted*** interpreter

then the interpreter has associated with it[6] a set of synthesized attributes values.

Type

```
(terminal, [attribute]) -> interpreter
```

Usage

interpreter_name = ***interpreted*** (*term_declaration*, [*list_of_attribute_values*])

where *term_declaration* could be any one of those defined above. The *list_of_attribute_values* could be zero or more attribute values of the form:

attribute_name ***attribute_value***

The type of ***attribute_value*** should be the same as that defined for **attribute_name** in the attribute definition section which appears at the beginning of the program. Whenever a terminal in the input string matches an ***interpreted*** definition then the attribute values associated with it become the synthesized attributes of the **interpreter_name**. Interpreters defined as ***interpreted*** do not have any inherited attributes, though they may have zero or more synthesized attributes.

---

[6] Inherited attributes cannot be defined for ***interpreted*** interpreters.

Example

```
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
attribute
::=
   LITERAL_VAL terminal
   | VAL num
special_symbols
=
  []
reserved_words
=
  ["one", "two", "three"]
type_of_terminals
=
  interpreted (RESERVED_WORD_TERM "one",[VAL 1])
  $excl_orelse
  interpreted (RESERVED_WORD_TERM "two",[VAL 2])
  $excl_orelse
  interpreted (RESERVED_WORD_TERM "three",[VAL 3])
```

Figure 3.6

```
server.uwindsor.ca

                Copyright Research Software Ltd. 1990


(300000 cells)
test3892.m
for help type /help
Miranda apply_interpreter type_of_terminals "one"
[([VAL 1],[])]
Miranda apply_interpreter type_of_terminals "two"
[([VAL 2],[])]
Miranda apply_interpreter type_of_terminals "three"
[([VAL 3],[])]
Miranda
```

Figure 3.7

In the first case, in Figure 4.7 above, since the first interpreter definition option matched input terminal **one**, **type_of_terminals** has a synthesized attribute **VAL** of type **num** whose value is **1**. Similar description applies for the second and third case.

3.2.6 Type of Interpreter Combinators

In W/AGE there are four combinators (functions) that may be used to define new interpreters in terms of other interpreters, namely, ***$excl_orelse***, ***$orelse***, and ***structure***, also refer to section 4.4.

3.2.6.1 $excl_orelse and $orelse

These two combinators can be used to define new interpreters, they have the following type:

```
interpreter -> interpreter -> interpreter
```

The above type definition means that these two combinators take two interpreters as arguments and return a new interpreter. As an example of their use, consider the following context-free grammar:

```
integer_or_identifier ::= integer_number
                        | identifier
```

The above context-free grammar would be coded[7] in W/AGE as:

---

[7] There are other ways to specify this in W/AGE.

```
integer_or_identifier
=
 literal INT_TERM
 $excl_orelse
 literal IDENTIFIER_TERM
```

OR

```
integer_or_identifier
=
 literal INT_TERM
 $orelse
 literal IDENTIFIER_TERM
```

Even though, ***$excl_orelse*** and ***$orelse*** have similar type and serve a common purpose there exists a subtle difference. Consider the following context-free grammar:

```
plus_expression ::= number
                  | number + plus_expression
```

where **number** is, say, any integer number. The following is a complete[8] W/AGE specification for the above grammar:

[8] No semantic actions are specified.

```
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
attribute
::=
   LITERAL_VAL terminal
special_symbols
=
 ['+']
reserved_words
=
 []
int_number
=
 literal INT_TERM
plus_sign
=
 uninterpreted (SPECIAL_SYMBOL_TERM "+")
plus_expression
=
 structure (s1 int_number)
 []
 $excl_orelse
 structure (s1 int_number ++ s2 plus_sign ++
            s3 plus_expression)
 []
```

Figure 3.8

After compiling the above program, the result of applying the function ***apply_interpreter*** to **plus_expression** with **2** as an argument is as shown below:

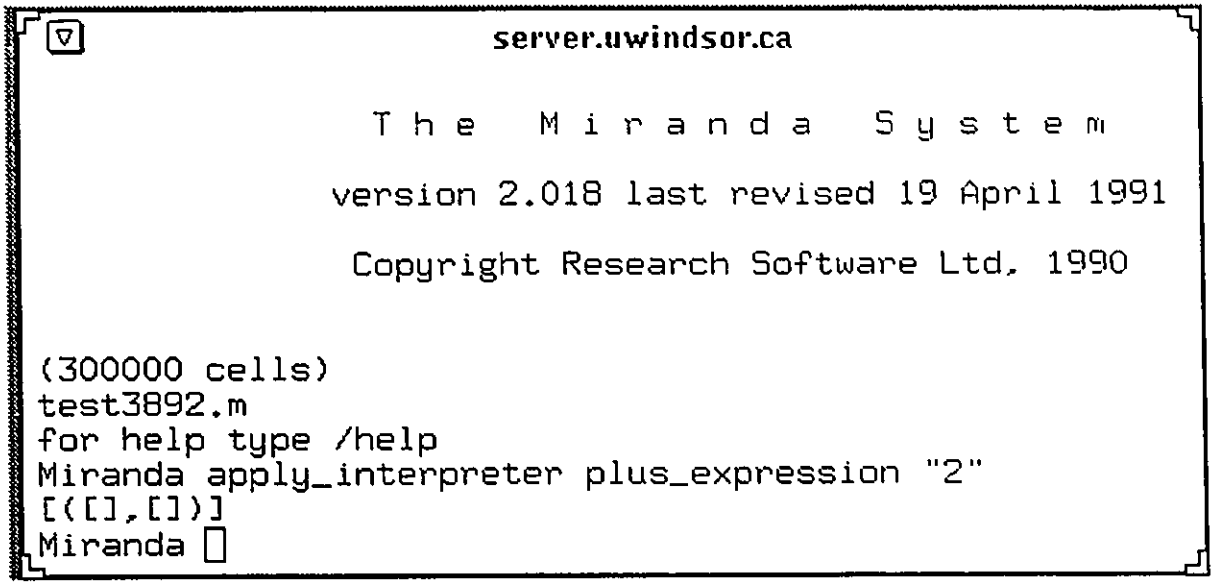

Figure 3.9

As shown in Figure 4.9 above, the terminal **2** was successfully accepted, denoted by [([],[])]. The result of applying the function ***apply_interpreter*** to **plus_expression** with **4 + 5** as arguments is shown below:

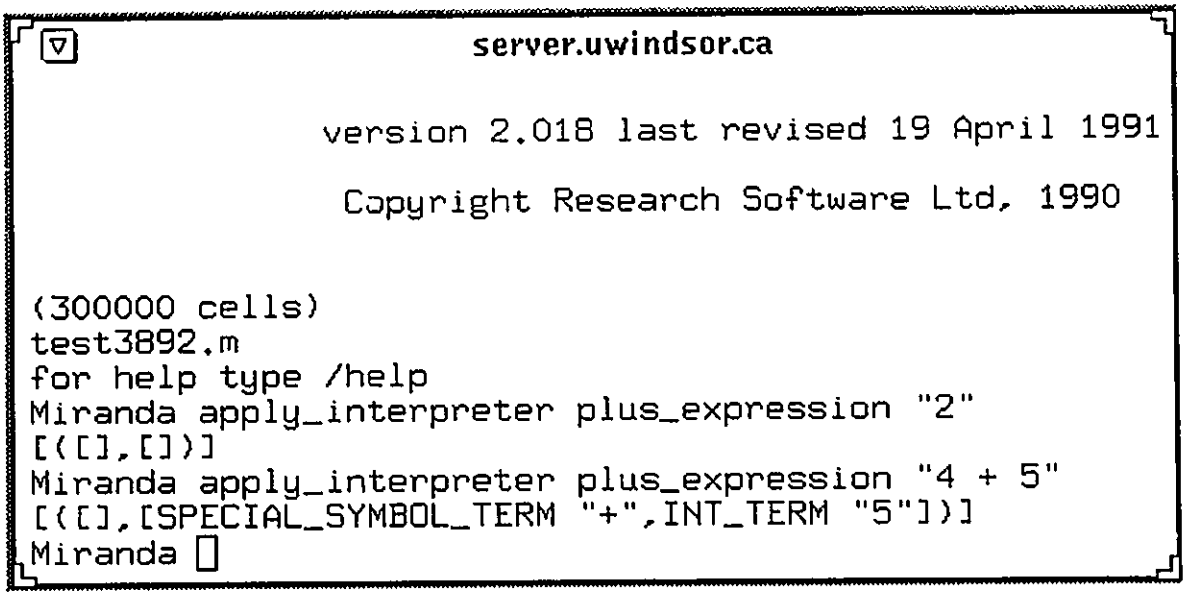

Figure 3.10

As shown in Figure 4.9 above, the terminals + and 5 failed to parse, but 4 was accepted. The reason for this failure is the way the interpreter combinator ***$excl_orelse*** works. Consider the following:

```
interpreter_name
=
 definition_1
 $excl_orelse
 definition_2
 $excl_orelse
 .....
 .....
 definition_n
```

If an interpreter definition has the above structure, then **definition_1** is first applied to the input string. If it succeeds, then **interpreter_name** succeeds; ***no matter if there are still input symbols to be consumed.*** If **definition_1** fails, only

then **definition_2** is applied to the input string. This process goes on until at least one definition succeeds, in which case the application of **interpreter_name** succeeds, or all of them fail, in which case the application of **interpreter_name** fails.

In our example case, the interpreter **int_number** is first applied to the input string **4 + 5**. This interpreter successfully consumes the first terminal, **4**, and succeeds, and as a result the application of the interpreter **plus_expression** succeeds even though there are remaining input symbols to be processed. The listing shown below is one way to get around this problem.

```
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
attribute
   LITERAL_VAL terminal
special_symbols
 ['+']
reserved_words
 []
int_number
 literal INT_TERM
plus_sign
 uninterpreted (SPECIAL_SYMBOL_TERM "+")
plus_expression
 structure (s1 int_number ++ s2 plus_sign ++
         s3 plus_expression)
 []
 $excl_orelse
 structure (s1 int_number)
 []
```

Figure 3.11

The listing above is similar to the previous one, except that the position of the two right hand side interpreter definition for **plus_expression** is interchanged.

In this case, the first right hand side interpreter definition for **plus_expression** is applied to 4 + 5. The interpreters **int_number** and **plus_sign** successfully consume the terminals 4 and +, respectively. The interpreter **plus_expression** is then recursively applied to the remaining terminal, namely, 5. Now, as it is self-explanatory, the application of the first right hand side interpreter definition for **plus_expression** to 5 will fail and as a result the second right hand side interpreter definition is applied, which succeeds. As a result, the recursive call made to **plus_expression** succeeds, which results in the success of the initial application of **plus_expression** to the input string 4 + 5.

The other way to get around the problem is to use the other interpreter combinator, *viz.*, ***$orelse***. The only difference between ***$excl_orelse*** and ***$orelse*** is that the latter parses the input string to be processed in ***all*** possible ways. The above example could be rewritten using ***$orelse*** as shown below:

```
%insert <local/header_for_WAGE_VERSION_2_RELEASE_0.m>
attribute
::=
   LITERAL_VAL terminal
special_symbols
=
 ['+']
reserved_words
=
 []
int_number
=
 literal INT_TERM
plus_sign
=
 uninterpreted (SPECIAL_SYMBOL_TERM "+")
plus_expression
=
 structure (s1 int_number)
 []
 $orelse
 structure (s1 int_number ++ s2 plus_sign ++
          s3 plus_expression)
 []
```

Figure 3.12

The result of applying the function ***apply_interpreter*** to **plus_expression** and the input string **4 + 5** is shown below:

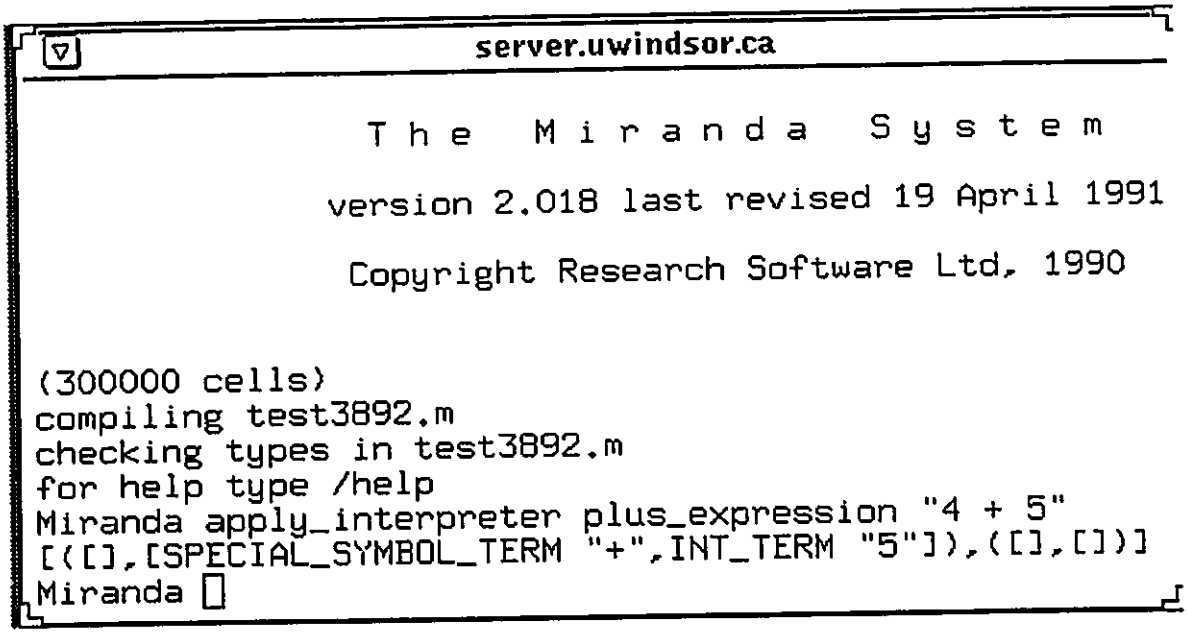

Figure 3.13

As shown in the figure above, the string was parsed in two different ways. The application of the first right hand side interpreter definition for **plus_expression** to the input string failed, while the application of the second right hand side interpreter definition succeeded.

The combinator **$excl_orelse** is used to improve efficiency but care should be taken in the order in which alternatives are listed in a production so that correct parses are not missed.

3.2.6.2 structure

This function (interpreter combinator) takes in a list of interpreters and a list

of semantic rules[9] and creates a new interpreter.

[9] The list of semantic may be empty.

Type

```
structure :: list_of_numbered_interpreters ->
                    list_of_attribute_rules ->
                    interpreters
where,
list_of_numbered_interpreters == [(num, interpreter)]
list_of_attribute_rules == [(num, att_id, att_function
                            , [att_id])]
where,
att_id == ((num, att_direction), att_type)
att_function == [attribute] -> attribute
att_type == [char]
att_direction ::= UP | DOWN
```

Usage

interpreter_name = *structure* (**s1 interpreter_name**$_1$ ++ ... ++ s_n **interpreter_name**$_n$)

[list_of_semantic_rules]

where each semantic rule in the **list_of_semantic_rules** specifies the dependency of the value of a synthesized or inherited attribute as a function of the value of other synthesized or inherited attributes.

In W/AGE, there are three types of semantic rules:

- **i_rule**: this is an initialization rule and is used when the value of an attribute needs to be initialized.

• **c_rule**: this is a copy rule, and is used when the value of an attribute is to be simply copied to another attribute; the name and type of the two attributes should be the same.

• **a_rule**: this is an application rule and is used when the value of an attribute is a function of one or more than one attributes.

Example

Figure 3.12 illustrates the use of the interpreter combinator, ***structure***.

APPENDIX 4
WAGE-ed User's Manual

§ 4.1 Basics

4.1.1 Generating WAGE-ed

As mentioned in section 5.3, there are two versions of **WAGE-ed**, *viz.*, guarded and unguarded editors. Guarded editors support left recursive interpreter definitions, whereas unguarded editors do not support left recursive interpreter definitions. Each version has five editors, the editors are distinguished from one another depending upon their level of versatility.

In order to generate an editor from an editor specification, type

sgen —o *output_file* —kernel ATO *input_file*

where *output_file* is an executable editor and *input_file* is the file containing the editor specification.

Note that the specification for **WAGE-ed** has been compiled using the ATO kernel. It will not compile using the ORDERED kernel, which is the default kernel used for compiling SSL specification, and works only for the ordered subclass of attribute grammars. Attribute grammar specification for **WAGE-ed** is unordered, therefore the specification will not compile with the new version of the Synthesizer Generator *viz.* version 4.0 because it supports only ORDERED kernel. However, the ATO kernel will be included in future versions of the Synthesizer Generator.

The editor for W/AGE available on our system is an unguarded **wage_editor** and was generated using the command,

sgen —o **xwageedit** —kernel ATO **wage_editor.ssl**

4.1.2 Invoking WAGE-ed

In order to invoke **WAGE-ed** on our system, type:

xwageedit*[filename]*

where the optional *filename* is the name of a file previously saved as ***structured*** or ***attribute*** form using **WAGE-ed.** If *filename* does not exist, then the program to be currently edited will be held in the buffer named *filename*.

xwageedit does not support left-recursive interpreter definitions, prohibits development of syntactically incorrect programs and performs various checking on attributes values and attribute definitions.

4.1.3 Exiting out of WAGE-ed

To exit from **WAGE-ed,** select ***exit*** from the **Edit** option or use ^C.

4.1.4 Saving Buffers

In order to save a program, select ***write-named-file*** from the **File** option. If the program is to be edited later, then it must be saved either in **structure** or **attributed** form. If the program is to be saved as text for compilation, then choose **text** form.

Note that a program saved as text *cannot* be re-edited using **WAGE-ed** later. Therefore, it is advisable to save a program in **text** and **structure/attributed** form.

§ 4.2 Using WAGE-ed

4.2.1 File Inclusion Section

In order to enter the header file, select <*path*>. Since no transformations are listed for <*path*>, text can be entered. In general, text can be entered from the keyboard if no transformations, listed in the help pane, are associated with a placeholder. The <*path*> placeholder can only be refined with alphanumeric characters, starting with an alphabet and may contain embedded underscores. The last pathname component may have a **.m** extension. Each component of the pathname must be entered one at a time. For example, in order to enter **local/header_for_WAGE_VERSION_2_RELEASE_0.m** after selecting <*path*> and keying in **local**, to enter **header_for_WAGE_VERSION_2_RELEASE_0.m** we have to get another <*path*> placeholder. To get one, select ***forward-sibling-with-optionals*** from the **Cursor** menu option or hit return key twice. If additional files are to be included, the template for entering pathname can be introduced by highlighting **%insert** which selects the entire declaration and then entering a sequence of return keys until a **%insert** template is obtained or by selecting ***forward-sibling-with-optionals*** from the **Cursor** menu option.

In order to delete any path component, highlight it and then choose ***delete-selection*** from the **Edit** menu option.

4.2.2 Attribute Declaration Section

Attribute names and their type can be declared by selecting *<attribute_name>* and *<attribute_type>* placeholders respectively. The *<attribute_name>* placeholder has no associated transformations, and it can only be refined by entering capital alphabetic characters, starting with an alphabet and may contain embedded underscores. In order to delete an attribute name, highlight it and then choose ***delete-selection*** from the **Edit** menu option.

The *<attribute_type>* placeholder has transformations associated with it. In order to refine it, select one of the choices from the help pane. User defined types can be entered by selecting ***User_Defined_Type*** from the help pane. In order to delete an attribute type, highlight it and then choose ***delete-selection*** from the **Edit** menu option.

Additional templates for declaring attribute names and their types can be obtained in one of the following ways:

• by clicking the left mouse button between the space separating the name and the type, this selects the entire declaration and then using ***forward-sibling-with-optionals*** or ***backward-sibling-with-optionals*** from the **Cursor** menu option,

• by clicking the left mouse button between the space separating the name and the type, this selects the entire declaration and then entering a sequence of return keys until a template for entering attribute name and type appears, or

• by highlighting the attribute type and then using ***forward-sibling-with-optionals*** from the **Cursor** menu option.

In order to delete an entire single attribute declaration *i.e.* an attribute name and its type, click the left mouse button between the space separating the name and the type, this selects the entire declaration and then use ***delete-selection*** from the **Edit** menu option.

4.2.3 Reserved Word Declaration Section

In order to declare reserved words, click the left mouse button on the keyword **reserved_words**, this selects the entire reserved word declaration section. Now select ***insert_reserved_words*** from the help pane, this introduces the *<reserved_word>* placeholder to enter a reserved word. A reserved word can be any word consisting of alphanumeric characters, starting with an alphabet and may contain underscores.

Additional placeholders for entering reserved words can be obtained in one of the following ways:

• if there is only a single declaration, then select the entire declaration by clicking the left mouse button on the keyword **reserved_words** and then choose

forward-with-optionals from the **Cursor** menu option,

• if there is more than one declaration, then select a declaration, and then choose ***forward-with-optionals*** from the **Cursor** menu option,

• if there is more than one declaration, then select a declaration, except the first one, and then choose ***backward-with-optionals*** from the **Cursor** menu option,

• by clicking the left mouse button on a declaration this selects the declaration and then entering a sequence of return keys until a *<reserved_word>* placeholder appears.

In order to delete a reserved word declaration, highlight it and then choose ***delete-selection*** from the **Edit menu** option.

4.2.4 Special Symbol Declaration Section

In order to declare reserved words, click the left mouse button on the keyword **special_symbols**, this selects the entire special symbols declaration section. Now select ***insert_spec_symbols*** from the help pane, this introduces the *<spec_symbols>* placeholder to enter a special symbol. A special symbol can be any character, other than alphanumeric characters.

Additional placeholders for entering special symbols can be obtained in one of the following ways:

• if there is only a single declaration, then select the entire declaration by clicking the left mouse button on the keyword **special_symbols** and then choose

forward-with-optionals from the **Cursor** menu option,

• if there is more than one declaration, then select a declaration, and then choose ***forward-with-optionals*** from the **Cursor** menu option,

• if there is more than one declaration, then select a declaration, except the first one, and then choose ***backward-with-optionals*** from the **Cursor** menu option,

• by clicking the left mouse button on a declaration this selects the declaration and then entering a sequence of return keys until a <*spec_symbols*> placeholder appears.

In order to delete a special symbol declaration, highlight it and then choose ***delete-selection*** from the **Edit menu** option.

4.2.5 Interpreter Definition Section

An interpreter in W/AGE may be one of **interpreted, uninterpreted, literal, recognised, guarded,** or **structured** type. A template for defining interpreter can be obtained by selecting <*interpreter*> placeholder and then transforming it into one of the above mentioned types by making a selection from the help pane.

In order to delete an entire interpreter definition select the entire definition by clicking on the equal sign and then using ***delete-selection*** from the **Edit** option. Templates for defining additional interpreters can be obtained, by selecting a previously entered interpreter definition (click on the equal sign) and then using

forward-sibling-with-optionals or ***backward-sibling-with-optionals***, this action will introduce an <*interpreter*> placeholder.

An interpreter defined as **literal** interpreter, has automatically associated with it a synthesized attribute **LITERAL_VAL** of user-defined type **terminal**.

An interpreter defined as an **interpreted** interpreter, may have one or more than one synthesized attributes associated with it.

An interpreter defined as an **uninterpreted** interpreter, has no attributes associated with it.

An interpreter defined as a **recognised** interpreter, has no semantic rules associated with it. The interpreter definition should be non-left-recursive.

An interpreter defined as a **guarded** interpreter is similar to a **structured** interpreter except that it has a *guard* (a **recognised** interpreter) and it can have left-recursive productions. The guard is applied to the input data, if it terminates only then the left-recursive definitions following the guard are applied to the input data.

An interpreter defined as a **structured** interpreter, may have one or more synthesized or inherited attributes associated with it. Semantic rules for productions can be entered by clicking on the keyword **structure** associated with the production, this introduces the <*semantic_rule*> placeholder. The <*semantic_rule*> placeholder can be transformed, by making a sequence of selections from the

help pane, into a template for entering an attribute equation for a synthesized or inherited attribute.

An attribute equation can be deleted by clicking on the rule number, this selects it, and then using ***delete-selection***. Templates for entering additional attribute equations can be introduced by first introducing the *<semantic_rule>* placeholder. To introduce the *<semantic_rule>* placeholder select a previously entered semantic rule and then use ***forward-sibling-with-optionals*** or ***backward-sibling-with-optionals.***

The attribute equations for inherited attributes must come before the attribute equations for synthesized attributes in the set of semantic rules associated with a production.

When an *<interprtr_defn>* placeholder is refined by clicking on **show** listed in the help pane, it gets transformed as shown below

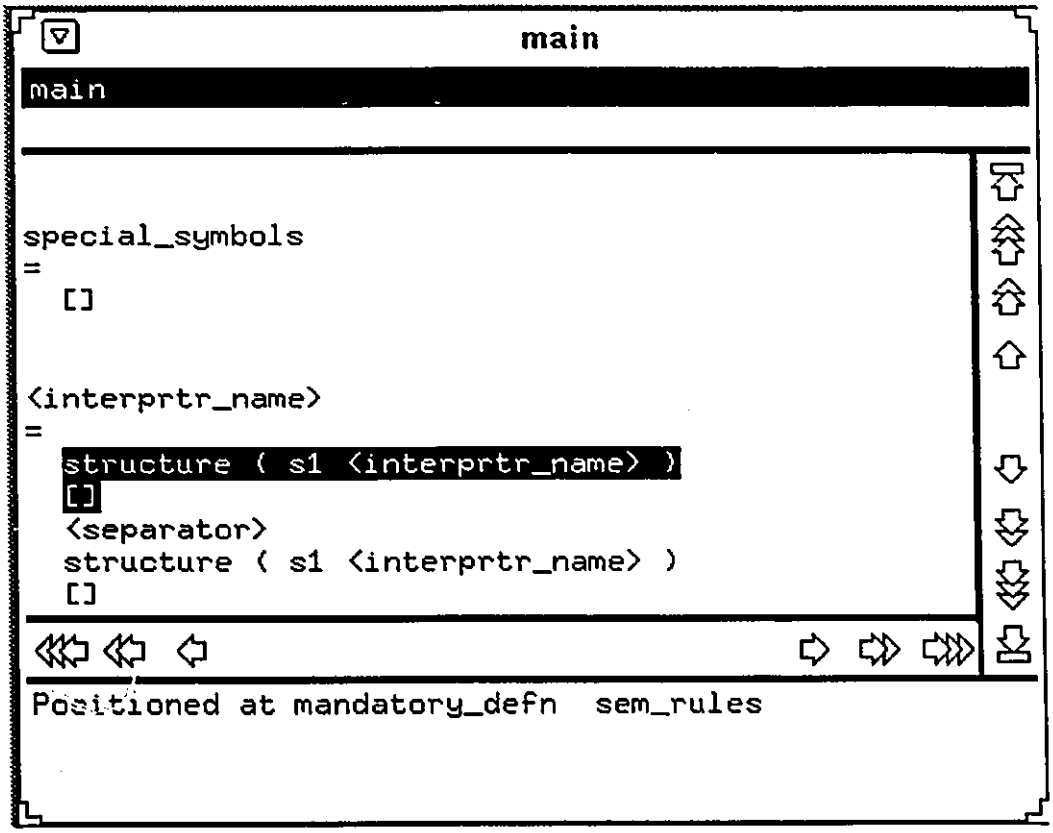

Figure 4.1

If the right hand side of an interpreter consists of only one production, then in order to get rid of the *<separator>* placeholder and the second structure template click on the *<separator>* placeholder which will select it and then click back on the immediately preceding structure definition. The following two figures illustrate this operation.

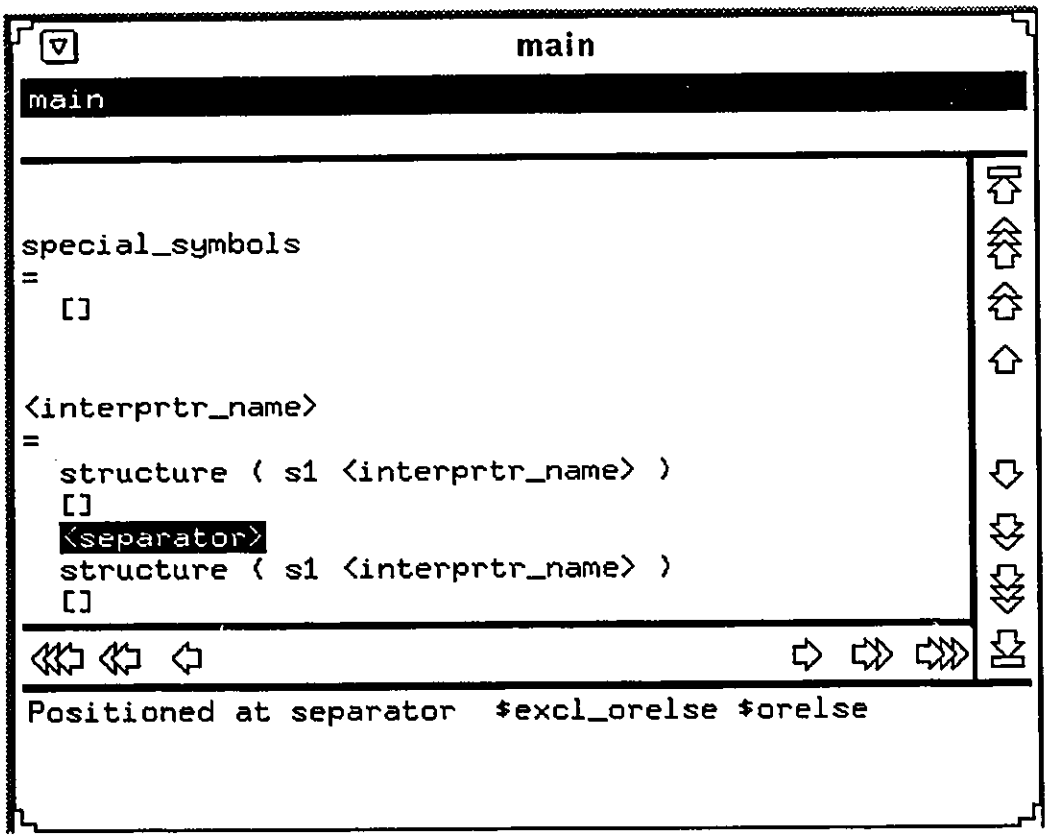
main
main
special_symbols
=
[]
<interprtr_name>
=
structure (s1 <interprtr_name>)
[]
<separator>
structure (s1 <interprtr_name>)
[]
Positioned at separator $excl_orelse $orelse

Figure 4.2

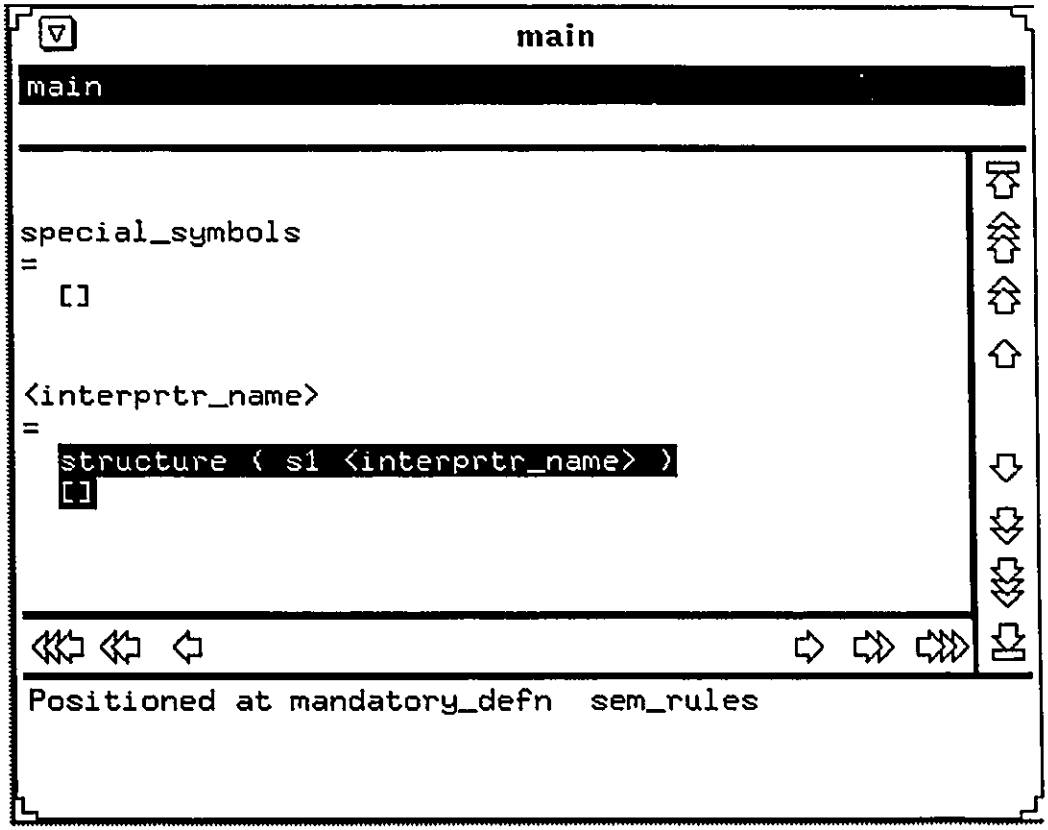

Figure 4.3

Suppose, we need additional structure templates: select the keyword **structure** and then use ***forward-sibling-with-optionals*** or ***backward-sibling-with-optionals*** or enter a series of returns. In order to delete a right hand side production and its associated semantic rules select it by clicking on the associated **structure** keyword and then use ***delete-selection***.

The <*separator*> placeholder can be refined by making a selection from the help pane. A refined <*separator*> placeholder can be deleted, by selecting it and

then using *delete-selection*.

§ 4.3 Editor Commands

4.3.1 Executing Commands

In **WAGE-ed**, commands can be executed in the following four ways:

- A command may be bound to a sequence of keystrokes.
- A command may be selected from the pop-up menu.
- A command name may be typed on the command line of a window.

In order to enter a command name on the command line, the **execute-command**, bound to TAB, must be invoked. Certain commands may need some extra parameters. The parameters are entered using a *parameter form* which appears as soon as the command is executed. After the parameters have been entered they are passed to the command by executing **start-command**. A command may be canceled at any time while entering its parameter by executing **cancel-command**. The following table contains a summary for executing commands through command line.

Command	Control Keys	Description
execute-command *name*	TAB *name*	execute the command *name*
illegal-operation	ESC-^G, ^G, ^X^G	cancel any incomplete command key-binding or partial entry on the command line
start-command	ESC-s	initiates execution of a command with parameters contained in the parameter form
cancel-command	ESC-c	cancel execution of a command awaiting completion of its parameter form
execute-monitor-command *command-line*	^X!	*command-line* is passed to UNIX to be executed as a command.
repeat-command	ESC-r	repeat the most recently initiated command. If the command had parameters then use the same parameters
return-to-monitor		call the UNIX shell

4.3.2 Transforms

If the current selection has any transformation(s) associated with it, then selection of this option will bring down a pull-down menu which will list all the legal transformations. These transformations are also listed at the bottom of the screen. The user has the choice of selecting a transformation either from the pull-down menu or making a selection[10] from the bottom of the screen.

If the current selection does not have any transformation(s) associated with it, then no pull-down menu will appear. In this case, **WAGE-ed** expects the user to key in text from the keyboard in order to refine the current selection.

---

[10] All selections in WAGE-ed are made by placing the mouse arrow on the text to be selected and then clicking the left mouse button.

4.3.3 Edit Commands

Command	Control Keys	Description
apropos *keyword*	ESC-?	lists all commands containing the keyword as a syllable or as a prefix of a syllable
text-capture		captures the text of the current selection into a text buffer. It fails if the current selection prohibits text entry
undo	^X^U	deletes the contents of a text buffer
cut-to-clipped	^W	moves the current selection to the buffer CLIPPED
copy-to-clipped	ESC-^W	copies the current selection to the buffer CLIPPED
paste-from-clipped	^Y	moves the contents of the CLIPPED buffer to the current selection, which must be a placeholder
copy-from-clipped	ESC-^Y	copies the contents of the CLIPPED buffer to the current selection ,which must be a placeholder

copy-text-from-clipped	ESC-^T	copies the contents of the CLIPPED buffer, as text, into a text buffer at the current selection immediately preceding the charater selection
delete-selection	^K	moves the contents of the current selection to the buffer DELETED

4.3.4 Cursor Commands

Command	Control Keys	Description
ascend-to-parent	ESC-\	changes the selection to the closest enclosing resting place
forward-preorder	^N	changes the selection to the next resting place in forward preorder traversal of the abstract syntax tree, does not stop at optional placeholders
forward-with-optionals	^M	changes the selection to the next resting place in forward preorder traversal of theabstract syntax tree, stops at optional placeholder

forward-sibling	ESC-^N	bypasses all resting place, without stopping at optional placeholders, contained within the current selection and advance to the next sibling in forward preorder traversal of the abstract syntax tree. If there is no next sibling, ascend to the enclosing resting place and advance to its next sibling
forward-sibling-with-optionals	ESC-^M	similar to forward-sibling, but stops at optional placeholders
backward-preorder	^P	changes the selection to the previous resting place in forward preorder traversal of the abstract syntax tree, does not stop at optional placeholders
backward-with-optionals	^H	similar to backward-preorder, but stops at optional placeholders
backward-sibling	ESC-^P	bypasses all resting places, without stopping at optional placeholders, contained within the current selection and advances to the previous sibling in forward preorder traversal of the abstract syntax tree. If there is no next sibling, then ascend to the enclosing resting place and advance to its next sibling

backward-sibling-with-optionals	ESC-^B	similar to backward-sibling, but stops at optional placeholders
beginning-of-file	ESC-<	changes the selection to the root of the abstract syntax tree
end-of-file	ESC->	changes the selection to the rightmost resting place in the abstract syntax tree
selection-to-top	ESC-!	scrolls the current selction, with respect to the window, so that it appears at the topmost line of the window

4.3.5 File Commands

Command	**Control Keys**	**Description**
read-file *filename*	^X^F, ^X^R	reads in a file previously saved in text, structured, or attributed form
visit-file *filename*	^X^V	reads a named file into a buffer with the same name. If a buffer already exists with that name, then a new buffer is created with that name suffixed by some integer *i*
insert-file *filename*	^X^I	replaces the current selection with the contents of *filename*

write-current-file	^Xs	writes the contents/value of the current buffer in the format associated with the buffer to its associated file
write-named-file *filename format*	^X^W	writes the contents of the current buffer to *filename* in the *format* specified. The default is **attributed**, other formats are **structure** and **text**
write-selection-to-file *filename format*		writes the current selection to *filename* in the *format* specified
write-modified-files	^X^M	writes the contents of every modified buffer to its associated file in the current format associated with the buffer
write-file-exit	^Xf	exits the editor after performing an operation similar to write-modified-files
write-attribute *attribute-name filename*		writes *attribute-name* of the current selection to *filename* in textual format

4.3.6 Buffer Commands

Commands	Control Keys	Description

list-buffers	^X^B	lists all buffers and their associated properties
switch-to-buffer *buffer-name*	^Xb	places the *buffer-name* in the current window. If no such buffer exist, then create a new buffer
new-buffer *buffer-name phylum*		creates a new buffer, in the current window, called as *buffer-name* and is initialized with the placeholder term of *phylum*. If *buffer-name* already exist, then the command is invalid

4.3.7 Window Commands

Command	Control Keys	Description
split-current-window	^X2	creates a new window and displays the same buffer in both the windows
delete-other-windows	^X1	deletes all windows, except the current one
delete-window	^Xd	deletes the current window

4.3.8 Search Command

Command	Control Keys	Description

search-forward *text*	ESC-^F	searches forward from the current selection for *text* in preorder traversal of the tree and wraps around to the root after reaching the end of the object
search-backward *text*	ESC-^R	searches backward from the current selection for *text* in preorder traversal of the tree and wraps around to the rightmost leaf and continues searching after reaching the root of the object

4.3.9 Optional Commands

Command	Control Keys	Description
set-parameters		allows the modification of global editor parameters
help-off		reduces the size of the help pane to zero
help-on		resets the size of the help pane to the default size
enlarge-help	ESC-^Xz	increases the size of the help pane of the current window by a line

shrink-help	ESC-^X^Z	reduces the size of the help pane of the current window by a line
show-copyright		displays copyright information

§ 4.4 Error Messages

When an attribute grammar specification is developed using **WAGE-ed**, it might be annotated with various warning messages. These warning messages arise if the program being edited violates any constraints. The warning messages serve to notify the user that an error exists in the program and may result in compilation errors, although **WAGE-ed** does not constrain the user in correcting them as soon as they appear.

An explanation about the cause of each warning message is given below.

Inconsistent Synthesized Attribute Definition

When the left hand side nonterminal has associated with it more than one definition (productions), then semantic rules associated with each production **must pass the same set of synthesized attributes up the left hand side nonterminal.** If this constraint is violated, then the program is annotated with **{INCOMP. SYNH. ATTR.}** warning message.

Undefined Attributes, Special Symbols and Reserved Words

If the attribute grammar specification being developed using **WAGE-ed** makes use of attributes, special symbols and reserved words which are not declared in attribute, special symbol, reserved word declaration section respectively, then **WAGE-ed** displays **{UNDEFINED}** warning message at locations in the program where the undeclared attribute, special symbol or reserved word is used.

Misuse of Reserved Word

If an interpreter is defined as an uninterpreted or interpreted interpreter to recognize a particular identifier in the input string, but if that identifier is predeclared as a reserved word then the interpreter definition is annotated with **{RESERVED WORD TERM}** message, notifying the user that the identifier is declared as a reserved word in the reserved word declaration section.

Multiple Attribute Definition

If an attribute is multiply defined in the attribute declaration section or if the semantic rules associated with a production contains more than one attribute equation synthesizing the same attribute up the left hand side nonterminal or inheriting the same attribute down to one of the right hand side nonterminal, then the program is annotated with the message **{ATTR. MULTIPLY DEFN.}**.

Left Recursive Definitions

If an *unguarded* **WAGE-ed** is used to develop an attribute grammar specification and if the definition for a structured interpreter contains left recursion, then the interpreter name (left hand side nonterminal) is juxtaposed with the message **{DEFN. CONTAINS LEFT RECURSION}.**

Multiple Interpreter Definition

If more than one interpreter definitions contain the same left hand side nonterminal name, then the interpreter name is juxtaposed with the message **{INTERPRETER ALREADY DEFINED}.**

Missing Inherited Attribute Equation

If a semantic rule associated with a production makes use of an inherited attribute value but if that attribute value is not available at the left hand side nonterminal, say x, of a production, then every occurrence of the nonterminal x used in the right hand side definition of all structured interpreter definitions are juxtaposed with the message **{MISSING INH. ATTR. EQN}.**

This notifies the user that the semantic rules associated with the production containing the message require an attribute equation for an inherited attribute.

Note: This warning message might still be there even if the attribute equation passes down an inherited attribute to a nonterminal (interpreter) appearing on

the right hand side of a production. This happens if attribute equation(s) for synthesized attribute(s) are defined before the attribute equation(s) for inherited attribute(s) in the collection of semantic rules associated with the production. To get rid of this make sure that **attribute equation(s) for inherited attribute(s) are defined before the attribute equation(s) for synthesized attribute(s) in the collection semantic rules.**

Type Mismatch

If a semantic rule is either an initialization or copy rule, then the name of the attribute on the left hand side of the **EQ** symbol must be the same as name on the right hand side or *vice-versa*. If this constraint is violated, then the semantic rule which violates this constraint is annotated with the message **{TYPE MISMATCH}**.

Circular Synthesized Attribute Definition

If the value of an attribute being synthesized up the left hand side nonterminal is a function of itself, then such an erroneous semantic rule is annotated with the message **{CIRC. ATTR. DEFN.}**.

Use of Undefined Synthesized Attribute Value

Reference in a semantic rule to a synthesized attribute value which has not been synthesized up an interpreter causes **WAGE-ed** to annotate the semantic rule which contains the reference with the message **{ERROR}**.

Reference to Undefined Interpreters

If a reference is made to an interpreter name , in a semantic rule, which does not appear on the right hand side of an interpreter definition then the definition is annotated with the message **{UNDEFINED INTR.}**.

Use of Undefined Guards

If an interpreter is used as a guard in the definition of a guarded interpreter, but has not been defined as a non-left-recursive **recognised** interpreter, then the interpreter name is juxtaposed with the message **{NOT RECOGNISED INTR.}**.

APPENDIX 5
WAGE-ed Source Code

Due to space limitation, only the source code for **unguarded** *wage_editor* (fully-fledged editor for W/AGE) is included here. Source code for other editor specification can be obtained from the School of Computer Science.

```
/*****************************************************************
*                                                                *
*                  Declaration of Lexemes                        *
*                                                                *
*****************************************************************/

NUMBER       : < "num" >
CHARACTER    : < "char" >
BOOLEAN      : < "bool" >
ANY          : < "any" >
EMPTY_TYPE   : < "()" >
START        : < "[(" >
END          : < ")]" >
SEPARATOR    : < "$excl_orelse" >
             | < "$orelse" >
RESWORDTERM  : < "RESERVED_WORD_TERM" >
SPECSYMBTERM : < "SPECIAL_SYMBOL_TERM" >
INTTERM      : < "INT_TERM" >
REALTERM     : < "REAL_TERM" >
ANYTERM      : < "ANY_TERM" >
IDENTTERM    : < "IDENTIFIER_TERM" >
INTEGER      : < [0-9]+ >
FLOAT        : < [0-9]+\.[0-9]+ >
ATTRIBUTE    : < [A-Z][A-Z_]*[A-Z]+ >
SYMBOL       : < [^a-zA-Z0-9] >
ALPHANUMERIC : < [a-z][a-z_0-9]+ >
RESWORD      : < [a-zA-Z][a-zA-Z_0-9]+ >
DIRNAME      : < [a-zA-Z]+[a-zA-Z0-9_]* >
FILENAME     : < [a-zA-Z]+[a-zA-Z0-9_]*\.m >
TERM         : < [\[\]{}a-zA-Z0-9:;"'^`!@#$%&*()_+=|,<>./?\\\-\^\ ]+ >
WHITESPACE   : < [\ \t\n] >
```

```
/*****************************************************************
*                                                                *
*              Attribute Type Definition                         *
*                                                                *
*****************************************************************/

ATTRIB              : AttribNil()
                    | AttribName(ATTRIBUTE)
                    ;

list SYNH_ATTR_LIST;
SYNH_ATTR_LIST      : SynhAttrListNil()
                    | SynhAttrDecl(SYNH_DECL  SYNH_ATTR_LIST)
                    ;

SYNH_DECL           : SynhDecl(ALPHANUMERIC  ATTRIB)
                    ;

list LIST_INTERPRTRS;
LIST_INTERPRTRS     : ListInterprtrsNil()
                    | ListInterprtrs(ALPHANUMERIC  LIST_INTERPRTRS)
                    ;

list  ATTRIB_LIST;
ATTRIB_LIST         : AttribListNil()
                    | AttribList(ATTRIB ATTRIB_LIST)
                    ;

list LOC_LIST_INTERPRTRS;
LOC_LIST_INTERPRTRS   : LocListInterprtrsNil()
                      | LocListInterprtrs(PAIR  LOC_LIST_INTERPRTRS)
                      ;

PAIR                : Pair(INT  ALPHANUMERIC)
                    ;

list LR_INTERPRTR_NAMES;
LR_INTERPRTR_NAMES  : LrInterprtrNamesNil()
                    | LrInterprtrNames(ALPHANUMERIC
                                       LR_INTERPRTR_NAMES)
                    ;

list RES_WORD_LIST;
RES_WORD_LIST       : ResWordListNull()
                    | ResWordListPair(STR  RES_WORD_LIST)
                    ;

list SPEC_SYMB_LIST;
SPEC_SYMB_LIST      : SpecSymbListNull()
                    | SpecSymbListPair(spec_symb  SPEC_SYMB_LIST)
                    ;

list INH_ATTR_LIST;
INH_ATTR_LIST       : InhAttrListNil()
                    | InhAttrList(INH_ATTR  INH_ATTR_LIST)
                    ;

INH_ATTR            : InhAttr(ALPHANUMERIC  ATTRIB_LIST)
                    ;

list LIST_ATTR_INT_PAIR;
LIST_ATTR_INT_PAIR  : ListAttrIntPairNil()
                    | ListAttrIntPair(ATTR_INT_PAIR
                                      LIST_ATTR_INT_PAIR)
                    ;

ATTR_INT_PAIR       : AttrIntPairNil()
                    | AttrIntPair(ATTRIB INT)
                    ;
```

```
/*****************************************************************
*                                                                *
*    Declaration of Phylum's Inh. and Synh. Attr.                *
*                                                                *
*****************************************************************/

list_of_attr_decl  ( synthesized ATTRIB_LIST attr_name_list;);

attr_decl_list ( synthesized ATTRIB_LIST attr_name_list;);

attr_decl      ( synthesized ATTRIB attr;);

attr_name      ( synthesized ATTRIB attr; );

decl_of_res_words ( synthesized RES_WORD_LIST res_word_table; );

list_of_res_words ( inherited RES_WORD_LIST  res_word_list;
                    synthesized RES_WORD_LIST res_word_table; );

res_word ( synthesized STR  resword; );

decl_of_spec_symbols ( synthesized SPEC_SYMB_LIST spec_symb_table; );

list_of_spec_symbols ( inherited SPEC_SYMB_LIST  spec_symb_list;
                       synthesized SPEC_SYMB_LIST spec_symb_table; );

spec_symb ( synthesized spec_symb  specsymb; );

decl_of_interpreters ( inherited SYNH_ATTR_LIST synh_list;
                       inherited LIST_INTERPRTRS intrprtr_list;
                       inherited INT int_inh;
                       /* INH */
                       inherited INH_ATTR_LIST inh_symb_table; );

types_of_interprtr (  inherited LIST_INTERPRTRS intrprtr_list;
                      synthesized ALPHANUMERIC name_of_interprtr;
                      synthesized ATTRIB_LIST  attr_list;
                      inherited INT int_inh;
                      synthesized INT int_synh;
                      /* INH */
                      synthesized ATTRIB_LIST u_inh_attr_list;);

an_interprtr_defn ( synthesized ATTRIB_LIST attr_list;
                    synthesized ALPHANUMERIC name_of_interprtr;
                    inherited LIST_INTERPRTRS intrprtr_list; );

interpreter_defn ( synthesized ATTRIB_LIST attr_list; );

definition ( synthesized ATTRIB_LIST attr_list; );

res_key_word ( synthesized STR key_word; );

spec_symb_key_word ( synthesized STR key_word; );

int_key_word ( synthesized STR key_word; );

real_key_word ( synthesized STR key_word; );

ident_key_word ( synthesized STR key_word; );

any_term_key_word ( synthesized STR key_word; );

op_list_of_interpreter_defn ( inherited ATTRIB_LIST attr_list; );

op_decl ( synthesized ATTRIB_LIST attr_list; );

list_of_intr_attr_decl ( synthesized ATTRIB_LIST attr_list; );

intr_attr_decl ( synthesized ATTRIB attr; );

an_uninterprtr_defn ( synthesized ALPHANUMERIC name_of_interprtr;
                      inherited LIST_INTERPRTRS intrprtr_list;  );

ident_str ( synthesized STR resword; );

spec_ident_str ( synthesized STR resword; );
```

```
literal_interprtr_defn  { synthesized  ALPHANUMERIC name_of_interprtr;
                          synthesized  ATTRIB_LIST attr_list;
                          inherited LIST_INTERPRTRS intrprtr_list; };

literal_decl    { synthesized ATTRIB_LIST attr_list; };

structured_defn          { inherited LIST_INTERPRTRS intrprtr_list;
                           synthesized ALPHANUMERIC name_of_interprtr;
                           synthesized ATTRIB_LIST attr_list;
                           inherited INT int_inh;
                           synthesized INT int_synh;
                           /*  INH */
                           synthesized ATTRIB_LIST u_inh_attr_list; };

interprtr_name  { inherited LIST_INTERPRTRS intrprtr_list;
                  synthesized ALPHANUMERIC name_of_interprtr; };

interprtr_defn  { inherited LIST_INTERPRTRS intrprtr_list;
                  synthesized ATTRIB_LIST attr_list;
                  synthesized LR_INTERPRTR_NAMES lr_interprtr_names;
                  inherited INT int_inh;
                  inherited INT frac_inh;
                  /* INH */
                  synthesized ATTRIB_LIST u_inh_attr_list; };

mandatory_defn { synthesized ATTRIB_LIST  mand_attr_list;
                 inherited LIST_INTERPRTRS intrprtr_list;
                 synthesized ALPHANUMERIC name_of_interprtr;
                 inherited INT int_inh;
                 inherited INT frac_inh;
                 synthesized INT frac_synh;
                 /*  INH */
                 inherited ATTRIB_LIST d_inh_attr_list;
                 synthesized ATTRIB_LIST u_inh_attr_list; };

list_of_rhs_interprtr_names { inherited LIST_INTERPRTRS intrprtr_list;
                              synthesized LOC_LIST_INTERPRTRS loc_list_intrprtrs;
                              synthesized ALPHANUMERIC name_of_interprtr;
                              inherited INT num;
                              /*  INH  */
                              inherited LIST_ATTR_INT_PAIR laip; };

rhs_interprtr_name { synthesized ALPHANUMERIC name_of_interprtr; };

list_of_semantic_rules { inherited ATTRIB_LIST inh_attr_list;
                         synthesized ATTRIB_LIST synh_attr_list;
                         inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
                         inherited INT int_inh;
                         inherited INT frac_inh;
                         synthesized INT frac_synh;
                         /* INH */
                         inherited ATTRIB_LIST d_inh_attr_list;
                         synthesized ATTRIB_LIST u_inh_attr_list;
                         synthesized LIST_ATTR_INT_PAIR laip; };

semantic_rule { synthesized ATTRIB attr;
                inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
                inherited INT int_inh;
                inherited INT frac_inh;
                /*  INH  */
                inherited ATTRIB_LIST d_inh_attr_list;
                synthesized ATTRIB_LIST u_inh_attr_list;
                synthesized ATTR_INT_PAIR aip; };

init_rule { synthesized ATTRIB attr;
            inherited INT int_inh;
            inherited INT frac_inh;
            inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
            /*  INH  */
            synthesized ATTR_INT_PAIR aip; };

copy_rule { synthesized ATTRIB attr;
```

```
            inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
            inherited INT int_inh;
            inherited INT frac_inh;
            /*  INH  */
            inherited ATTRIB_LIST d_inh_attr_list;
            synthesized ATTRIB_LIST u_inh_attr_list;
            synthesized ATTR_INT_PAIR aip; };

appl_rule { synthesized ATTRIB attr;
            inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
            inherited INT int_inh;
            inherited INT frac_inh;
            /*  INH  */
            inherited ATTRIB_LIST d_inh_attr_list;
            synthesized ATTRIB_LIST u_inh_attr_list;
            synthesized ATTR_INT_PAIR aip; };

list_of_rhs_attr_type { inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
                        synthesized ATTRIB_LIST circ_lhs_attr_list;
                        /* INH */
                        inherited ATTRIB_LIST d_inh_attr_list;
                        synthesized ATTRIB_LIST u_inh_attr_list; };

synh_rhs_attr_type { inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
                     synthesized ATTRIB attr;
                     synthesized ATTRIB circ_lhs_attr; };

synh_lhs_attr_type { synthesized ATTRIB attr; };

inh_lhs_attr_type { inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
                    /* INH */
                    synthesized ATTR_INT_PAIR aip;
                    synthesized ATTRIB inh_lhs_attr; };

inh_rhs_attr_type { synthesized ATTRIB inh_rhs_attr;
                    /* INH */
                     };

inh_or_synh_type { synthesized ATTRIB attr;
                   inherited LOC_LIST_INTERPRTRS loc_list_intrprtrs;
                   synthesized ATTRIB circ_lhs_attr;
                   /*  INH  */
                   inherited ATTRIB_LIST d_inh_attr_list;
                   synthesized ATTRIB_LIST u_inh_attr_list;
                   synthesized ATTRIB inh_rhs_attr; };

interprtr_number { synthesized INT num; };

optional_list_of_defns { inherited   ATTRIB_LIST inhopt_attr_list;
                         inherited LIST_INTERPRTRS intrprtr_list;
                         synthesized LR_INTERPRTR_NAMES lr_interprtr_names;
                         inherited INT int_inh;
                         inherited INT frac_inh;
                         /* INH */
                         inherited ATTRIB_LIST d_inh_attr_list;
                         synthesized ATTRIB_LIST u_inh_attr_list; };

op_defn { synthesized ATTRIB_LIST attr_list;
          inherited LIST_INTERPRTRS intrprtr_list;
          synthesized ALPHANUMERIC name_of_interprtr;
          inherited INT int_inh;
          inherited INT frac_inh;
          synthesized INT frac_synh;
          /* INH */
          inherited ATTRIB_LIST d_inh_attr_list;
          synthesized ATTRIB_LIST u_inh_attr_list; };
```

```
/*****************************************************************
*                                                                *
*                    SSL Specification for W/AGE                 *
*                                                                *
*****************************************************************/


/*  Abstract  Syntax */
root  prog;
prog          :  Prog(header_files  list_of_attr_decl decl_of_res_words
                      decl_of_spec_symbols  decl_of_interpreters)
              ;


/*****************************************************************
*                                                                *
*         SSL Specification for File Inclusion Section in W/AGE  *
*                                                                *
*****************************************************************/

list header_files;
header_files : HeaderNil()
             | HeaderFiles(path header_files)
             ;

list path;
path         : PathNil()
             | PathDecl(pathname path)
             ;

pathname     : PathNameNil()
             | PathCapName(ATTRIBUTE)
             | PathAlphaNumName(ALPHANUMERIC)
             | PathResWordName(RESWORD)
             | PathFileName(FILENAME)
             ;

/*       Semantic  Rules     */

prog  : Prog{decl_of_interpreters.synh_list = SynhAttrListNil;
             decl_of_interpreters.intrprtr_list = ListInterprtrsNil;
             local ATTRIB_LIST attr_defn_table;
             attr_defn_table = list_of_attr_decl.attr_name_list;
             decl_of_interpreters.int_inh = 1;
             local RES_WORD_LIST res_word_table;
             local SPEC_SYMB_LIST spec_symb_table;
             res_word_table = decl_of_res_words.res_word_table;
             spec_symb_table = decl_of_spec_symbols.spec_symb_table;
             /*  INH  */
             decl_of_interpreters.inh_symb_table = InhAttrListNil;};


/*  Unparsing Rules */
style Normal, Keyword, Placeholder, Error;
prog         : Prog[^ ::= ^ "%n%n%n" ^ "%n%n%n" ^ "%n%n%n" ^ "%n%n%n" ^]
             ;

header_files : HeaderFiles[ @ ::=  "%S(Keyword:%%insert%S) <" ^ ">" ["%n"] @ ]
             ;

path         : PathNil[@ ::=]
             | PathDecl[@ ::= ^ ["/"] @]
             ;

pathname     : PathNameNil[@ ::= "%S(Placeholder:<path>%S)"]
             | PathCapName[@ ::= ^]
             | PathAlphaNumName[@ ::= ^]
             | PathResWordName[@ ::= ^]
             | PathFileName[@ ::= ^]
             ;
```

```
/*  Entry  Declarations  */

Pathname { synthesized pathname val; };
pathname ^ Pathname.val;

/*  Parsing Rules */

Pathname ::= (ATTRIBUTE)       {Pathname.val    PathCapName(ATTRIBUTE);}
         |   (ALPHANUMERIC)    {Pathname.val    PathAlphaNumName(ALPHANUMERIC);}
         |   (RESWORD)         {Pathname.val    PathResWordName(RESWORD);}
         |   (FILENAME)        {Pathname.val    PathFileName(FILENAME);}
         ;
```

```
/*****************************************************************
*                                                                *
*        SSL Specification for Attribute Definition Section in W/AGE        *
*                                                                *
*****************************************************************/


/*  Abstract  Syntax */

list_of_attr_decl  : AttrDecls(attr_decl_list)
                   ;

list attr_decl_list;
attr_decl_list     :  AttrDeclNil()
                   |  ListofAttrDecl(attr_decl  attr_decl_list)
                   ;

attr_decl          :  AttrDeclPair(attr_name  attr_type)
                   ;

attr_name          : AttrNameNil()
                   | AttrName(ATTRIB)
                   ;

attr_type          : AttrTypeNil()
                   | AttrTypeName(typename)
                   | AttrTupleType(tuple_type)
                   | AttrListType(list_type)
                   | AttrAlphaNumType(user_defined_type)
                   | AttrFunctionType(func_type)
                   ;

list tuple_type;
tuple_type         : TupleTypeNil()
                   | TupleTypeList(attr_type  tuple_type)
                   ;

list list_type;
list_type          : ListTypeNil()
                   | ListofListType(attr_type  list_type)
                   ;

func_type          : FuncType(attr_type  attr_type)
                   ;

user_defined_type  : UserDefinedTypeNil()
                   | UserDefinedType(ALPHANUMERIC)
                   ;

typename           : NumType(number)
                   | CharType(character)
                   | BoolType(boolean)
                   | EmptyType(empty)
                   ;

number             : NumVal(NUMBER)
                   ;

character          : CharVal(CHARACTER)
                   ;

boolean            : BoolVal(BOOLEAN)
                   ;

empty              : EmptyVal(EMPTY_TYPE)
                   ;

/*     Semantic  Rules   */
list_of_attr_decl  : AttrDecls{list_of_attr_decl$1.attr_name_list =
                              attr_decl_list.attr_name_list;};

attr_decl_list     : AttrDeclNil{attr_decl_list$1.attr_name_list = AttribListNil;}
                   | ListofAttrDecl{attr_decl_list$1.attr_name_list =
                                       InsertAttrName(attr_decl.attr,
                                        attr_decl_list$2.attr_name_list);
                                     local STR error;
                                     error = (ChkAttribList(attr_decl.attr,
                                              attr_decl_list$2.attr_name_list)) ?
```

```
                                        "{MULTIPLE DEFN.}" : "";};
attr_decl : AttrDeclPair{attr_decl.attr   attr_name.attr;};

attr_name : AttrNameNil{attr_name$1.attr   AttribNil; }
          | AttrName{attr_name$1.attr   ATTRIB; };


/*   Definition  of  Functions in Semantic Rules  */
ATTRIB_LIST InsertAttrName(ATTRIB nl, ATTRIB_LIST sl) {

         (nl == AttribNil) ? sl
         : with(sl) {
             AttribListNil : nl::sl,
             AttribList(ATTRIB, tl) : (nl    ATTRIB) ? sl :
                                 ATTRIB::InsertAttrName(nl, tl)
           }
};

BOOL ChkAttribList(ATTRIB nl, ATTRIB_LIST sl) {

       (nl == AttribNil) ? false
       : with(sl) {
          AttribListNil : false,
          AttribList(ATTRIB, tl) : (nl -- ATTRIB) ? true :
                         ChkAttribList(nl, tl)
        }
};


/*       Unparsing  Rules  */
list_of_attr_decl  : AttrDecls[^ ::= "%S(Keyword:attribute%S)%n:: %t%t%n" ^]
                   ;

attr_decl_list  :  AttrDeclNil[^ ::= "%b%b%n"]
                |  ListofAttrDecl[@ ::= ^  error["%n| "] @ ]
                ;

attr_decl       :  AttrDeclPair [^ ::= ^ " " ^ ]
                ;

attr_name       : AttrNameNil[@ ::= "%S(Placeholder:<attribute_name>%S)"]
                | AttrName[@ ::= ^]
                ;

attr_type       : AttrTypeNil[@ ::= "%S(Placeholder:<attribute_type>%S)"]
                | AttrTypeName[@ ::= ^]
                | AttrTupleType[@ : "(" ^ ")"]
                | AttrListType[@ : "[(" ^ ")]"]
                | AttrAlphaNumType[@ ::= ^]
                | AttrFunctionType[@ ::= ^]
                ;

tuple_type      : TupleTypeNil[@ ::=]
                | TupleTypeList[@ ::= ^ [","] @]
                ;

list_type       : ListTypeNil[@ ::=]
                | ListofListType[@ ::= ^ [","] @]
                ;

func_type       : FuncType[^ : "(" ^ " -> " ^")"]
                ;

user_defined_type : UserDefinedTypeNil[@ ::= "<user_defined_type>"]
                  | UserDefinedType[^ ::= ^]
                  ;

typename        : NumType[^ ::= ^]
                | CharType[^ ::= ^]
                | BoolType[^ ::= ^]
                | EmptyType[ ^ ::= ^]
                ;

number          : NumVal[^ ::= ^]
                ;

character       : CharVal[^ ::= ^]
                ;
```

```
boolean         : BoolVal[^ :: ^]
                ;

empty           : EmptyVal[^ :: ^]
                ;

/*   Entry  Declarations  */

Attr_name { synthesized attr_name  val; };
attr_name ^ Attr_name.val;
Attr_type { synthesized  attr_type val; };
attr_type ^ Attr_type.val;
List_type { synthesized list_type val; };
list_type ^ List_type.val;
Tuple_type { synthesized tuple_type val; };
tuple_type ^ Tuple_type.val;
Typename { synthesized typename val; };
typename ^ Typename.val;
Number { synthesized number val; };
number ^ Number.val;
Character { synthesized character val; };
character ^ Character.val;
Boolean { synthesized boolean val; };
boolean ^ Boolean.val;
Empty {synthesized empty val;};
empty ^ Empty.val;
User_defined_type { synthesized user_defined_type val; };
user_defined_type ^ User_defined_type.val;

/*  Parsing Declarations */
Attr_name   ::=  (ATTRIBUTE) {Attr_name.val =
                              AttrName(AttribName(ATTRIBUTE));}
            ;
Attr_type   ::= (Typename) {Attr_type.val =
                              AttrTypeName(Typename.val);}
            | ('('Tuple_type')') {Attr_type.val =
                                  AttrTupleType(Tuple_type.val);}
            | (START List_type END)  {Attr_type.val =
                                     AttrListType(List_type.val);}
            ;
User_defined_type ::= (ALPHANUMERIC) {User_defined_type.val =
                                      UserDefinedType(ALPHANUMERIC);}
                  ;
Tuple_type  ::= (Attr_type) {Tuple_type.val =
                             TupleTypeList(Attr_type.val,
                                           TupleTypeNil());}
            |   (Attr_type ',' Tuple_type) {Tuple_type$1.val =
                             TupleTypeList(Attr_type.val,
                                           Tuple_type$2.val);}
            ;
List_type   ::= (Attr_type) {List_type.val =
                             ListofListType(Attr_type.val,
                                            ListTypeNil());}
            |   (Attr_type ',' List_type) {List_type$1.val =
                             ListofListType(Attr_type.val,
                                            List_type$2.val);}
            ;
Typename    ::= (Number)    {Typename.val = NumType(Number.val);}
            |   (Character) {Typename.val = CharType(Character.val);}
            |   (Boolean)   {Typename.val = BoolType(Boolean.val);}
            |   (Empty)     {Typename.val = EmptyType(Empty.val);}
            ;
Number      ::= (NUMBER)  {Number.val = NumVal(NUMBER);}
            ;
Character   ::= (CHARACTER) {Character.val = CharVal(CHARACTER);}
            ;
Boolean     ::= (BOOLEAN)  {Boolean.val = BoolVal(BOOLEAN);}
            ;
Empty       ::= (EMPTY_TYPE) {Empty.val = EmptyVal(EMPTY_TYPE);}
            ;
```

```
/*  Transformation  Declarations  */
transform  attr_type  on "num"  <attr_type> :
                                AttrTypeName(NumType(NumVal("num")));
transform  attr_type  on "char" <attr_type> :
                                AttrTypeName(CharType(CharVal("char")));
transform  attr_type  on "bool" <attr_type> :
                                AttrTypeName(BoolType(BoolVal("bool")));
transform attr_type   on "Empty_type"  <attr_type> :
                                AttrTypeName(EmptyType(EmptyVal("()")))
                                ;
transform  attr_type  on "Tuple_type"  <attr_type> :
                                AttrTupleType([tuple_type]);
transform  attr_type  on "List_type"  <attr_type> :
                                AttrListType(<list_type>)
                                ;
transform attr_type  on "Function_type" <attr_type> :
                                AttrFunctionType(<func_type>)
                                ;
transform attr_type on "User_Defined_Type" <attr_type> :
                                AttrAlphaNumType(<user_defined_type>)
                                ;
```

```
/*****************************************************************
*                                                                *
*    SSL Specification for Reserved Word Declaration Section in W/AGE    *
*                                                                *
*****************************************************************/

/*  Abstract  Syntax  */
decl_of_res_words : ResWordList(list_of_res_words)
                  ;
optional list list_of_res_words;
list_of_res_words : ResWordListNil()
                  | ResWordDecl(res_word  list_of_res_words)
                  ;
res_word          : ResWordNil()
                  | ResWord(RESWORD)
                  | AlphaNum(ALPHANUMERIC)
                  | UpCase(ATTRIBUTE)
                  ;

/*   Semantic  Rules  */

decl_of_res_words : ResWordList{list_of_res_words.res_word_list =
                                                    ResWordListNull;
                                decl_of_res_words$1.res_word_table =
                                   list_of_res_words.res_word_table;};

list_of_res_words : ResWordListNil{list_of_res_words$1.res_word_table
                                        = ResWordListNull;}
                  | ResWordDecl{list_of_res_words$2.res_word_list =
                                  InsertResWordList(res_word.resword,
                                    list_of_res_words$1.res_word_list);
                                local STR error;
                                error = (ChkResWordList(res_word.resword,
                                     list_of_res_words$1.res_word_list)) ?
                                     "(MULTIPLE DEFN.)" : "";
                                list_of_res_words$1.res_word_table =
                                  InsertResWordList(res_word.resword,
                                      list_of_res_words$2.res_word_table);};

res_word  : ResWordNil{res_word$1.resword = ""; }
          | ResWord{res_word$1.resword = RESWORD; }
          | AlphaNum{res_word$1.resword = ALPHANUMERIC; }
          | UpCase{res_word$1.resword = ATTRIBUTE; }
          ;

/*   Definition  of  Function in Semantic  Rules  */
RES_WORD_LIST  InsertResWordList(STR nl, RES_WORD_LIST sl) {

         (nl == "") ? sl
         : with(sl) {
            ResWordListNull : nl::sl,
            ResWordListPair(STR, tl) : (STR == nl) ? sl
                                     : STR::InsertResWordList(nl, tl)
          }
};
BOOL ChkResWordList(STR nl, RES_WORD_LIST sl) {

         (nl == "") ? false
         : with(sl) {
            ResWordListNull : false,
            ResWordListPair(STR, tl) : (STR == nl) ? true
                                     : ChkResWordList(nl, tl)
          }
};


/*  Unparsing  Rules  */
decl_of_res_words : ResWordList[@ ::= "%S(Keyword:reserved_words%S)
                                       %n=%t%n" "[^"]"]
```

```
                ;

list_of_res_words : ResWordListNil[@ ::= "%b"]
                  | ResWordDecl[^ ::= "\"" ^ "\"" error [",%o"] @]
                  ;

res_word          : ResWordNil[@ ::= "%S(Placeholder:<reserved word>%S)"]
                  | ResWord[^ ::= ^]
                  | AlphaNum[^ ::= ^]
                  | UpCase[^ ::= ^]
                  ;

/*  Entry  Declaration */

Res_word { synthesized res_word val;};
res_word ^ Res_word.val;

/*  Parsing  Declarations */

Res_word ::= (RESWORD)       {Res_word.val = ResWord(RESWORD); }
           | (ALPHANUMERIC)  {Res_word.val = AlphaNum(ALPHANUMERIC); }
           | (ATTRIBUTE)     {Res_word.val = UpCase(ATTRIBUTE); }
           ;

/* Transformation Rules */

transform  decl_of_res_words  on "insert_reserved_word"
        ResWordList([list_of_res_words]) : ResWordList(<list_of_res_words>)
                                         ;
```

```
/*********************************************************************
*                                                                    *
*    SSL Specification for Special Symbol Declaration Section in W/AGE   *
*                                                                    *
*********************************************************************/


/*  Abstract  Syntax  */

decl_of_spec_symbols : SpecSymbList(list_of_spec_symbols)
                     ;

optional list list_of_spec_symbols;

list_of_spec_symbols : SpecSymbListNil()
                     | SpecSymbDecl(spec_symb  list_of_spec_symbols)
                     ;

spec_symb            : SpecSymbNil()
                     | SpecSymb(SYMBOL)
                     ;

/*   Semantic  Rules  */
decl_of_spec_symbols : SpecSymbList{list_of_spec_symbols.spec_symb_list =
                                                    SpecSymbListNull;
                                    decl_of_spec_symbols$1.spec_symb_table =
                                     list_of_spec_symbols.spec_symb_table;};

list_of_spec_symbols : SpecSymbListNil{list_of_spec_symbols$1.spec_symb_table =
                                        SpecSymbListNull;}
                     | SpecSymbDecl{list_of_spec_symbols$2.spec_symb_list =
                                    InsertSpecSymbList(spec_symb.specsymb,
                                     list_of_spec_symbols$1.spec_symb_list);
                                local STR error;
                                error = (ChkSpecSymbList(spec_symb.specsymb,
                                     list_of_spec_symbols$1.spec_symb_list)) ?
                                     "{MULTIPLE DEFN.}" : "";
                                list_of_spec_symbols$1.spec_symb_table =
                                   InsertSpecSymbList(spec_symb.specsymb,
                                     list_of_spec_symbols$2.spec_symb_table);};

spec_symb  : SpecSymbNil{spec_symb$1.specsymb = SpecSymbNil; }
           | SpecSymb{spec_symb$1.specsymb = SpecSymb(SYMBOL); }
           ;


/*    Definition  of  Function  in Semantic  Rules   */
SPEC_SYMB_LIST  InsertSpecSymbList(spec_symb n1, SPEC_SYMB_LIST s1) {

         (n1 == SpecSymbNil) ? s1
         : with(s1) {
            SpecSymbListNull : n1::s1,
            SpecSymbListPair(SYMBOL, t1) : (SYMBOL == n1) ? s1
                                         : SYMBOL::InsertSpecSymbList(n1, t1)
          }
};

BOOL ChkSpecSymbList(spec_symb n1, SPEC_SYMB_LIST s1) {

         (n1 == SpecSymbNil) ? false
         : with(s1) {
            SpecSymbListNull : false,
            SpecSymbListPair(SYMBOL, t1) : (SYMBOL == n1) ? true
                                         : ChkSpecSymbList(n1, t1)
          }
};

/*  Unparsing  Rules  */
decl_of_spec_symbols : SpecSymbList[@ ::= "%S(Keyword:special_symbols%S)
                                          %n=%t%n" "["^"]"]
```

```
                    ;

list_of_spec_symbols : SpecSymbListNil[@ ::= "%b"]
                     | SpecSymbDecl[^ ::= "^" ^ "^" error [",%o"] @]
                     ;

spec_symb            : SpecSymbNil[@ ::= "%S(Placeholder:<spec symbols>%S)"]
                     | SpecSymb[^ ::= ^]
                     ;

/*  Entry  Declaration */

Spec_symb { synthesized spec_symb val;};
spec_symb   ^ Spec_symb.val;

/*  Parsing  Declarations */

Spec_symb ::= (SYMBOL) {Spec_symb.val = SpecSymb(SYMBOL);}
          ;

/*      Transformation    Rules     */

transform  decl_of_spec_symbols  on "insert_spec_symbols"
        SpecSymbList([list_of_spec_symbols]) :
                        SpecSymbList(<list_of_spec_symbols>)
```

```
/*****************************************************************
*                                                                *
*     SSL Specification for  Interpreter Definition Section in W/AGE     *
*                                                                *
*****************************************************************/

/*   Abstract  Syntax  */

list decl_of_interpreters;
decl_of_interpreters       : DeclofInterpretersNil()
                           | DeclofInterpreters(types_of_interprtr
                                                decl_of_interpreters)
                           ;

types_of_interprtr         : NilInterprtr()
                           | InterprtredInterprtr(an_interprtr_defn)
                           | UninterprtredInterprtr(an_uninterprtr_defn)
                           | StructuredInterprtr(structured_defn)
                           | LiteralInterprtr(literal_interprtr_defn)
                           ;

decl_of_interpreters :
          DeclofInterpreters{
         local SYNH_ATTR_LIST a;
                        a = InsertSynhList(types_of_interprtr.name_of_interprtr,
                                           types_of_interprtr.attr_list,
                                           decl_of_interpreters$1.synh_list);
                       decl_of_interpreters$2.synh_list =
                           InsertSynhList(types_of_interprtr.name_of_interprtr,
                                           types_of_interprtr.attr_list,
                                           decl_of_interpreters$1.synh_list);
                         types_of_interprtr.intrprtr_list =
                           decl_of_interpreters$1.intrprtr_list;
                         decl_of_interpreters$2.intrprtr_list =
                           ChkIntrprtrDef(types_of_interprtr.name_of_interprtr,
                           decl_of_interpreters$1.intrprtr_list) ?
                           decl_of_interpreters$1.intrprtr_list :
                           InsertLIstInterprtrs(types_of_interprtr.name_of_interprtr,
                                                decl_of_interpreters$1.intrprtr_list);
                         types_of_interprtr.int_inh =
                                           decl_of_interpreters$1.int_inh;
                         decl_of_interpreters$2.int_inh =
                                           types_of_interprtr.int_synh + 1;
                         /* INH */
                         decl_of_interpreters$2.inh_symb_table =
                                (ChkInhAttrList(types_of_interprtr.name_of_interprtr,
                                                decl_of_interpreters$1.inh_symb_table) &&
                                   (types_of_interprtr.u_inh_attr_list != AttribListNil))
                               ? decl_of_interpreters$1.inh_symb_table
                               : InsertSymbTable(types_of_interprtr.name_of_interprtr,
                                                 types_of_interprtr.u_inh_attr_list,
                                                 decl_of_interpreters$1.inh_symb_table);
                         local INH_ATTR_LIST h;
                         h = (ChkInhAttrList(types_of_interprtr.name_of_interprtr,
                                             decl_of_interpreters$1.inh_symb_table) &&
                                   (types_of_interprtr.u_inh_attr_list != AttribListNil))
                               ? decl_of_interpreters$1.inh_symb_table
                               : InsertSymbTable(types_of_interprtr.name_of_interprtr,
                                                 types_of_interprtr.u_inh_attr_list,
                                                 decl_of_interpreters$1.inh_symb_table);};

types_of_interprtr  : NilInterprtr(types_of_interprtr.attr_list = AttribListNil;
                                   types_of_interprtr.name_of_interprtr = "";
                                   types_of_interprtr$1.int_synh = 0;
                                   /* INH */
                                   types_of_interprtr$1.u_inh_attr_list = AttribListNil; )
                    | InterprtredInterprtr(types_of_interprtr.attr_list =
                                             an_interprtr_defn.attr_list;
                                           types_of_interprtr.name_of_interprtr =
```

```
                                    an_interprtr_defn.name_of_interprtr;
                                types_of_interprtr$1.int_synh  0;
                                an_interprtr_defn.intrprtr_list
                                    types_of_interprtr$1.intrprtr_list;
                        /* INH */
                        types_of_interprtr$1.u_inh_attr_list  AttribListNil;}
    | UninterprtredInterprtr{types_of_interprtr.attr_list
                                AttribListNil;
                            types_of_interprtr.name_of_interprtr
                               an_uninterprtr_defn.name_of_interprtr;
                            types_of_interprtr$1.int_synh  0;
                            an_uninterprtr_defn.intrprtr_list
                                types_of_interprtr$1.intrprtr_list;
                        /* INH */
                        types_of_interprtr$1.u_inh_attr_list  AttribListNil;}
    | StructuredInterprtr{structured_defn.intrprtr_list
                             types_of_interprtr$1.intrprtr_list;
                          types_of_interprtr$1.attr_list
                             structured_defn.attr_list;
                          types_of_interprtr$1.name_of_interprtr
                             structured_defn.name_of_interprtr;
                          types_of_interprtr$1.int_synh
                             structured_defn.int_synh;
                          structured_defn.int_inh
                             types_of_interprtr$1.int_inh;
                          /* INH */
                          types_of_interprtr$1.u_inh_attr_list
                             structured_defn.u_inh_attr_list;}
    | LiteralInterprtr{types_of_interprtr.attr_list
                          literal_interprtr_defn.attr_list;
                       types_of_interprtr.name_of_interprtr
                          literal_interprtr_defn.name_of_interprtr;
                       types_of_interprtr$1.int_synh  0;
                       literal_interprtr_defn.intrprtr_list
                          types_of_interprtr$1.intrprtr_list;
                    /* INH */
                    types_of_interprtr$1.u_inh_attr_list  AttribListNil;};

/*    Unparsing  Rule  */
decl_of_interpreters      : DeclofInterpretersNil[@ ::=]
                          | DeclofInterpreters[@ ::= ^ ["%b%n%n"] @]
                          ;

types_of_interprtr        : NilInterprtr[@ ::= "%t%n%S(Placeholder:<interpreter>%S)"]
                          | InterprtredInterprtr[^ ::= ^]
                          | UninterprtredInterprtr[^ ::= ^]
                          | StructuredInterprtr[^ ::= ^]
                          | LiteralInterprtr[^ ::= ^]
                          ;

/*   Transformation  Rules  */
transform types_of_interprtr  on "interpreted" <types_of_interprtr> :
                                  InterprtredInterprtr(<an_interprtr_defn>),
                              on "uninterpreted" <types_of_interprtr> :
                                  UninterprtredInterprtr(<an_uninterprtr_defn>),
                              on "structured" <types_of_interprtr> :
                                  StructuredInterprtr(<structured_defn>),
                              on "literal" <types_of_interprtr> :
                                  LiteralInterprtr(<literal_interprtr_defn>)
                              ;
```

```
/*********************************************************************
*       SSL  Specification for  INTERPRETED  interpreters            *
*********************************************************************/

/*  Abstract  Syntax  */
an_interprtr_defn          : InterpreterDecl(interprtr_name interpreter_defn)
                           ;

interpreter_defn           : InterpreterDefnNil()
                           | InterpreterDefn(definition op_list_of_interpreter_defn)
                           ;

definition                 : DefnDecl(term_decl list_of_intr_attr_decl)
                           ;

optional list list_of_intr_attr_decl;
list_of_intr_attr_decl     : ListofIntrAttrDeclNil()
                           | ListofIntrAttrDecl(intr_attr_decl list_of_intr_attr_decl)
                           ;

intr_attr_decl             : IntrAttrDecl(attr_name any_thing)
                           ;

optional list op_list_of_interpreter_defn;
op_list_of_interpreter_defn  : OpDeclNil()
                             | DeclList(op_decl op_list_of_interpreter_defn)
                             ;

op_decl                    : OpDecl(separator definition)
                           ;

separator                  : SeparatorNil()
                           | SeparatorType(SEPARATOR)
                           ;

term_decl                  : TermDeclNil()
                           | ResWordDefn(res_key_word res_word)
                           | SpecSymbDefn(spec_symb_key_word  spec_symb)
                           | IntTermDefn(int_key_word int_str)
                           | RealTermDefn(real_key_word real_str)
                           | IdentTermDefn(ident_key_word ident_str)
                           | AnyTermDefn(any_term_key_word any_str)
                           ;

res_key_word               : ResKeyWord(RESWORDTERM)
                           ;

spec_symb_key_word         : SpecSymbKeyWord(SPECSYMBTERM)
                           ;

int_key_word               : IntKeyWord(INTTERM)
                           ;

int_str                    : IntStrNil()
                           | IntStr(integer_val)
                           | AnyWildInt(any_wild_symb)
                           ;

integer_val                : IntegerValNil()
                           | IntegerVal(INTEGER)
                           ;

real_key_word              : RealKeyWord(REALTERM)
                           ;

real_str                   : RealStrNil()
                           | RealStr(real_val)
                           | AnyWildReal(any_wild_symb)
                           ;

real_val                   : RealValNil()
                           | RealVal(FLOAT)
                           ;

ident_key_word             : IdentKeyWord(IDENTTERM)
                           ;

ident_str                  : IdentdNil()
                           | AnyWildIdent(any_wild_symb)
                           | SpecIdent(spec_ident_str)
                           ;
```

```
spec_ident_str              : SpecIdentStrNil()
                            | Ident(RESWORD)
                            | IdentAlphaNum(ALPHANUMERIC)
                            | IdentUpCase(ATTRIBUTE)
                            ;

any_term_key_word           : AnyTermKeyWord(ANYTERM)
                            ;

any_str                     : AnyStrNil()
                            | AnyWildAnyStr(any_wild_symb)
                            | SpecAnyStr(spec_any_str)
                            ;

spec_any_str                : AnySpecSymb(SYMBOL)
                            | AnyResWord(RESWORD)
                            | AnyAlphaNum(ALPHANUMERIC)
                            | AnyCap(ATTRIBUTE)
                            | AnyInt(INTEGER)
                            | AnyReal(FLOAT)
                            ;

any_thing                   : AnyThingNil()
                            | AnyThingCap(ATTRIBUTE)
                            | AnyThingAlphaNum(ALPHANUMERIC)
                            | AnyThingResWord(RESWORD)
                            | AnyThingInt(INTEGER)
                            | AnyThingReal(FLOAT)
                            | AnyThingElse(TERM)
                            ;

any_wild_symb               : AnyWildSymb(ANY)
                            ;

/*   Semantic Rules  */
an_interprtr_defn  : InterpreterDecl{an_interprtr_defn$1.attr_list =
                                       interpreter_defn.attr_list;
                                     an_interprtr_defn$1.name_of_interprtr =
                                       interprtr_name.name_of_interprtr;
                                     interprtr_name.intrprtr_list =
                                       an_interprtr_defn$1.intrprtr_list;};

interpreter_defn   : InterpreterDefnNil{interpreter_defn$1.attr_list =
                                          AttrIbListNil;}
                   | InterpreterDefn{interpreter_defn$1.attr_list =
                                       definition.attr_list;
                                     op_list_of_interpreter_defn.attr_list =
                                       definition.attr_list;};

op_list_of_interpreter_defn : DeclList{op_list_of_interpreter_defn$2.attr_list =
                                         op_list_of_interpreter_defn$1.attr_list;
                                       local STR error;
                                       error = (ChkMandOptListDefns(op_decl.attr_list,
                                                  op_list_of_interpreter_defn$1.attr_list))
                                               ? "" : "(INCOMP. SYNH. ATTR.)";};

op_decl    : OpDecl{op_decl$1.attr_list = definition.attr_list;};

definition : DefnDecl{definition$1.attr_list = list_of_intr_attr_decl.attr_list;};

term_decl                  : ResWordDefn{local STR error;
                                         error = ((res_key_word.key_word == "RESERVED_WORD_TERM") &&
                                                    (res_word.resword != ""))
                                                 ? ChkResWordList(res_word.resword, (Proq.res_word_table))
                                                     ? "" : "(UNDEFINED)"
                                                 : "";}
                           | SpecSymbDefn{local STR error;
                                          error = ((spec_symb_key_word.key_word ==
                                                      "SPECIAL_SYMBOL_TERM") &&
                                                   (spec_symb.specsymb != SpecSymbNil))
                                                  ? ChkSpecSymbList(spec_symb.specsymb, (Proq.spec_symb_tab
                                                      ? "" : "(UNDEFINED)"
                                                  : "";}
```

```
                        | IdentTermDefn{local STR error;
                                      error   ChkResWordTable(ident_key_word.key_word,
                                                ident_str.resword, {Prog.res_word_table})
                                              ? "{RESERVED WORD TERM}" : "";};

ident_str               : IdentdNil{ident_str$1.resword = "";}
                        | AnyWildIdent{ident_str$1.resword = "";}
                        | SpecIdent{ident_str$1.resword = spec_ident_str.resword;}
                        ;
spec_ident_str          : SpecIdentStrNil{spec_ident_str$1.resword = "";}
                        | Ident{spec_ident_str$1.resword = RESWORD;}
                        | IdentAlphaNum{spec_ident_str$1.resword = ALPHANUMERIC;}
                        | IdentUpCase{spec_ident_str$1.resword = ATTRIBUTE;}
                        ;

res_key_word    : ResKeyWord{res_key_word$1.key_word = RESWORDTERM;};
spec_symb_key_word : SpecSymbKeyWord{spec_symb_key_word$1.key_word = SPECSYMBTERM;};
int_key_word    : IntKeyWord{int_key_word$1.key_word = INTTERM;};
real_key_word : RealKeyWord{real_key_word$1.key_word = REALTERM;};
ident_key_word : IdentKeyWord{ident_key_word$1.key_word = IDENTTERM;};
any_term_key_word : AnyTermKeyWord{any_term_key_word$1.key_word = ANYTERM;};

list_of_intr_attr_decl : ListofIntrAttrDeclNil{
                              list_of_intr_attr_decl$1.attr_list = AttribListNil;}
                       | ListofIntrAttrDecl{
                            list_of_intr_attr_decl$1.attr_list =
                              InsertAttrList(intr_attr_decl.attr,
                                list_of_intr_attr_decl$2.attr_list);
                           local STR error;
                           error = (ChkAttrList(intr_attr_decl.attr,
                                     list_of_intr_attr_decl$2.attr_list))
                                   ? "{ATTR. MULTIPLY DEFN.}" : "";};

intr_attr_decl       : IntrAttrDecl{local STR error;
                                    error = (ChkAttrDefnTable(attr_name.attr,
                                                              {Prog.attr_defn_table}))
                                            ? "" : "{UNDEFINED}";
                                     intr_attr_decl$1.attr = attr_name.attr;};

/*    Definition  of  Function in Semantic Rules  */
BOOL  ChkAttrDefnTable(ATTRIB n, ATTRIB_LIST sl) {

        (n == AttribNil) ? true
        : with(sl) {
           AttribListNil : false,
           AttribList(ATTRIB, tl) : (n == ATTRIB) ? true
                                    : ChkAttrDefnTable(n, tl)
         }
};

BOOL ChkResWordTable(STR k, STR s, RES_WORD_LIST sl) {

         ((k == "IDENTIFIER_TERM") && (s != ""))
         ? ChkResWordList(s, sl)
           ? true : false
         : false
};

/*  Unparsing  Rules */
an_interprtr_defn       : InterpreterDecl[^ : ^ "%n=%t%n" ^]
                        ;
interpreter_defn        : InterpreterDefnNil[@ ::= "%S(Placeholder:<interpreter_defn>%S)"]
                        | InterpreterDefn[@ ::=  ^ "%n" ^]
                        ;
```

```
definition                  : DefnDecl[@ ::  "%S(Keyword:interpreted%S) (" ^ " ,|" ^ "|)"]
                            ;

term_decl                   : TermDeclNil[@ ::  "%S(Placeholder:<term_decl>%S)"]
                            | ResWordDefn[@ : ^ " \"" ^ "\"" error]
                            | SpecSymbDefn[@ : ^ " \"" ^ "\"" error]
                            | IntTermDefn[@ : ^ " " ^]
                            | RealTermDefn[@ : ^ " " ^]
                            | IdentTermDefn[@ : ^ " " ^  error]
                            | AnyTermDefn[@ : ^ " " ^]
                            ;

res_key_word                : ResKeyWord[^ : ^]
                            ;

spec_symb_key_word          : SpecSymbKeyWord[^ : ^]
                            ;

int_key_word                : IntKeyWord[^ : ^]
                            ;

int_str                     : IntStrNil[@ ::  "%S(Placeholder:<int_str>%S)"]
                            | IntStr[^ ::= "\""^"\""]
                            | AnyWildInt[^ ::= ^]
                            ;

integer_val                 : IntegerValNil[@ ::  "<integer_value>"]
                            | IntegerVal[^ ::= ^]
                            ;

real_key_word               : RealKeyWord[^ : ^]
                            ;

real_str                    : RealStrNil[@ ::= "%S(Placeholder:<real_str>%S)"]
                            | RealStr[^ ::= "\""^"\""]
                            | AnyWildReal[^ ::= ^]
                            ;

real_val                    : RealValNil[@ ::= "<real_value>"]
                            | RealVal[^ ::= ^]
                            ;

ident_key_word              : IdentKeyWord[^ : ^]
                            ;

ident_str                   : IdentdNil[@ ::= "%S(Placeholder:<identifier>%S)"]
                            | SpecIdent[^ ::= "\""@"\""]
                            | AnyWildIdent[^ ::= @]
                            ;

spec_ident_str              : SpecIdentStrNil[@ ::= "<user_defined_ident>"]
                            | Ident[^ ::= ^]
                            | IdentAlphaNum[^ ::= ^]
                            | IdentUpCase[^ ::= ^]
                            ;

any_term_key_word           : AnyTermKeyWord[^ : ^]
                            ;

any_str                     : AnyStrNil[@ ::= "%S(Placeholder:<any_str>%S)"]
                            | AnyWildAnyStr[^ ::= ^]
                            | SpecAnyStr[^ ::= "\""^ "\""]
                            ;

spec_any_str                : AnySpecSymb[@ ::= ^]
                            | AnyResWord[@ ::= ^]
                            | AnyAlphaNum[@ ::= ^]
                            | AnyCap[@ ::= ^]
                            | AnyInt[@ ::= ^]
                            | AnyReal[@ ::=^]
                            ;

any_wild_symb               : AnyWildSymb[^ ::= ^]
                            ;

list_of_intr_attr_decl      : ListofIntrAttrDeclNil[@ ::=]
                            | ListofIntrAttrDecl[@ ::= ^ error [",%o"] @]
                            ;

intr_attr_decl              : IntrAttrDecl[^ ::= error @ " " @]
                            ;

op_list_of_interpreter_defn : OpDeclNil[@ ::=]
                            | DeclList[@ ::=  ^ error "%n" @]
```

```
                              ;
op_decl                       : OpDecl[" ::= " "%n" "]
                              ;

separator                     : SeparatorNil[@ ::= "%S(Placeholder:<separator>%S)"]
                              | SeparatorType[@ ::= "]
                              ;

any_thing                     : AnyThingNil[@ ::= "%S(Placeholder:<any_thing>%S)"]
                              | AnyThingCap[@ ::= "]
                              | AnyThingAlphaNum[@ ::= "]
                              | AnyThingResWord[@ ::= "]
                              | AnyThingInt[@ ::= "]
                              | AnyThingReal[@ ::= "]
                              | AnyThingElse[@ ::= "]
                              ;

/*  Entry  Declarations */
Separator {synthesized separator val;};
separator ^ Separator.val;
Integer_val {synthesized integer_val val;};
integer_val ^ Integer_val.val;
Real_val {synthesized real_val val;};
real_val ^ Real_val.val;
Spec_any_str {synthesized spec_any_str val;};
spec_any_str ^ Spec_any_str.val;
Spec_ident_str {synthesized spec_ident_str val;};
spec_ident_str ^ Spec_ident_str.val;
Any_thing {synthesized any_thing val;};
any_thing ^ Any_thing.val;

/*  Parsing  Declarations  */
Separator       ::= (SEPARATOR) {Separator.val = SeparatorType(SEPARATOR);}
                ;

Integer_val     ::= (INTEGER)   {Integer_val.val = IntegerVal(INTEGER);}
                ;

Real_val        ::= (FLOAT)     {Real_val.val = RealVal(FLOAT);}
                ;

Spec_any_str      ::= (INTEGER)       {Spec_any_str.val = AnyInt(INTEGER);}
                  |   (FLOAT)         {Spec_any_str.val = AnyReal(FLOAT);}
                  |   (SYMBOL)        {Spec_any_str.val = AnySpecSymb(SYMBOL);}
                  |   (RESWORD)       {Spec_any_str.val = AnyResWord(RESWORD);}
                  |   (ALPHANUMERIC)  {Spec_any_str.val = AnyAlphaNum(ALPHANUMERIC);}
                  |   (ATTRIBUTE)     {Spec_any_str.val = AnyCap(ATTRIBUTE);}
                  ;

Spec_ident_str    ::= (RESWORD)       {Spec_ident_str.val = Ident(RESWORD);}
                  |   (ALPHANUMERIC)  {Spec_ident_str.val = IdentAlphaNum(ALPHANUMERIC);}
                  |   (ATTRIBUTE)     {Spec_ident_str.val = IdentUpCase(ATTRIBUTE);}
                  ;

Any_thing       ::= (ATTRIBUTE)     {Any_thing.val = AnyThingCap(ATTRIBUTE);}
                |   (ALPHANUMERIC)  {Any_thing.val = AnyThingAlphaNum(ALPHANUMERIC);}
                |   (RESWORD)       {Any_thing.val = AnyThingResWord(RESWORD);}
                |   (INTEGER)       {Any_thing.val = AnyThingInt(INTEGER);}
                |   (FLOAT)         {Any_thing.val = AnyThingReal(FLOAT);}
                |   (TERM)          {Any_thing.val = AnyThingElse(TERM);}
                ;


/*  Transformation  Declarations  */
transform interpreter_defn on "show" <interpreter_defn> :
                                     InterpreterDefn(<definition>,
                                                     <op_list_of_interpreter_defn>)
                                                      ;

transform term_decl on "Reserved_Word_Decl" <term_decl>:
                       ResWordDefn(ResKeyWord("RESERVED_WORD_TERM"),<res_word>),
```

```
                    on "Special_Symb_Decl"  <term_decl>:
                        SpecSymbDefn(SpecSymcKeyWord("SPECIAL_SYMBOL_TERM"),
                                                        <spec_symb>),

                    on "Integer_Decl"       <term_decl>:
                        IntTermDefn(IntKeyWord("INT_TERM"),<int_str>),

                    on "Real_Decl"          <term_decl>:
                        RealTermDefn(RealKeyWord("REAL_TERM"),<real_str>),

                    on "Identifier_Decl"    <term_decl>:
                        IdentTermDefn(IdentKeyWord("IDENTIFIER_TERM"),<ident_str>),

                    on "AnyTerm_Decl"       <term_decl>:
                        AnyTermDefn(AnyTermKeyWord("ANY_TERM"),<any_str>)
                                                        ;

transform int_str on "integer"  <int_str> : IntStr(<integer_val>),
                  on "any"      <int_str> : AnyWildInt(AnyWildSymb("any"))
                                          ;

transform real_str on "real"    <real_str> : RealStr(<real_val>),
                   on "any"     <real_str> : AnyWildReal(AnyWildSymb("any"))
                                           ;

transform ident_str on  "identifier"  <ident_str> : SpecIdent(<spec_ident_str>),
                    on  "any"         <ident_str> : AnyWildIdent(AnyWildSymb("any"))
                                                  ;

transform any_str  on  "string"       <any_str>   : SpecAnyStr(<spec_any_str>),
                   on  "any"          <any_str>   : AnyWildAnyStr(AnyWildSymb("any"))
                                                  ;

transform separator on "$excl_orelse"   <separator> : SeparatorType("$excl_orelse"),
                    on "$orelse"        <separator> : SeparatorType("$orelse")
                                                    ;

transform definition on "Attribute Declarations"
                            DefnDecl(s, [list_of_intr_attr_decl]) :
                                    DefnDecl(s, <list_of_intr_attr_decl>);
```

```
/*************************************************************************
*        SSL Specification  for  UNINTERPRETED  interpreters             *
*************************************************************************/

/*  Abstract  Syntax  */

an_uninterprtr_defn      : UnInterpreterDecl(interprtr_name uninterprtr_defn)
                         ;

uninterprtr_defn   : UnInterprtrDefnNil()
                   | UnInterprtrDefn(undefinition op_list_of_uninterprtr_defn)
                   ;
undefinition       : UnDefnDecl(term_decl)
                   ;
optional list op_list_of_uninterprtr_defn;
op_list_of_uninterprtr_defn : UnOpDeclNil()
                            | UnDeclList(unop_decl op_list_of_uninterprtr_defn)
                            ;
unop_decl                   : UnOpDecl(separator undefinition)
                            ;

/*  Semantic Rules  */
an_uninterprtr_defn  : UnInterpreterDecl{an_uninterprtr_defn$1.name_of_interprtr =
                                          interprtr_name.name_of_interprtr;
                                        interprtr_name.intrprtr_list =
                                          an_uninterprtr_defn$1.intrprtr_list;};


/*  Unparsing  Rules */
an_uninterprtr_defn         : UnInterpreterDecl[^ : ^ "%n=%t%n" ^]
                            ;
uninterprtr_defn            : UnInterprtrDefnNil[@ :.= "%S(Placeholder:<interpreter_defn>%S)"]
                            | UnInterprtrDefn[@ ::=  ^ "%n" ^]
                            ;
undefinition                : UnDefnDecl[@ ::= "%S(Keyword:uninterpreted%S)  (" ^ ")"]
                            ;
op_list_of_uninterprtr_defn : UnOpDeclNil[@ ::=]
                            | UnDeclList[@ ::= ^ "%n" @]
                            ;
unop_decl                   : UnOpDecl[@ ::= ^ "%n" ^]
                            ;

/*  Transformation  Declarations  */

transform uninterprtr_defn on "show" <uninterprtr_defn> :
                                     UnInterprtrDefn(<undefinition>,
                                                     <op_list_of_uninterprtr_defn>)
                                                   ;
```

```
/*************************************************************************
*          SSL Specification for LITERAL interpreters                    *
*************************************************************************/

/*  Abstract  Syntax  */

literal_interprtr_defn      : LiteralInterpreterDecl(interprtr_name literal_decl)
                            ;

literal_decl        : LiteralDeclNil()
                    | LiteralDecl(keyword_decl op_list_of_literal_decl)
                    ;

optional list op_list_of_literal_decl;
op_list_of_literal_decl     : OpLiteralDeclNil()
                            | LiteralDeclList(op_literal_decl op_list_of_literal_decl)
                            ;

op_literal_decl           : OpLiteralDecl(separator keyword_decl)
                          ;


keyword_decl                : KeywordDeclNil()
                            | ResKeyWordDecl(res_key_word)
                            | SpecSymbKeyWordDecl(spec_symb_key_word)
                            | IntKeyWordDecl(int_key_word)
                            | RealKeyWordDecl(real_key_word)
                            | IdentKeyWordDecl(ident_key_word)
                            | AnyTermKeyWordDecl(any_term_key_word)
                            ;


/*  Semantic Rules  */
literal_interprtr_defn  : LiteralInterpreterDecl{literal_interprtr_defn$1.name_of_interprtr =
                                                    interprtr_name.name_of_interprtr;
                                                literal_interprtr_defn$1.attr_list =
                                                    literal_decl.attr_list;
                                                interprtr_name.interprtr_list =
                                                    literal_interprtr_defn$1.interprtr_list;};

literal_decl  : LiteralDeclNil{literal_decl$1.attr_list = AttribListNil;}
              | LiteralDecl{literal_decl$1.attr_list =
                            AttribList(AttribName("LITERAL_VAL"), AttribListNil);};


/*  Unparsing  Rules */
literal_interprtr_defn          : LiteralInterpreterDecl[" : " "%n%t%n" "]
                                ;

literal_decl        : LiteralDeclNil[@ ::= "%S(Placeholder:<interpreter_defn>%S)"]
                    | LiteralDecl[@ ::= " "%n" "]
                    ;

op_list_of_literal_decl     : OpLiteralDeclNil[@ ::=]
                            | LiteralDeclList[@ ::= " "%n" @]
                            ;

op_literal_decl           : OpLiteralDecl[@ ::= " "%n" "]
                          ;


keyword_decl                : KeywordDeclNil[@ ::= "%S(Placeholder:<literal_declaration>%S)"]
                            | ResKeyWordDecl[@ : "%S(Keyword:literal %S)" "]
                            | SpecSymbKeyWordDecl[@ : "%S(Keyword:literal %S)" "]
                            | IntKeyWordDecl[@ : "%S(Keyword:literal %S)" "]
                            | RealKeyWordDecl[@ : "%S(Keyword:literal %S)" "]
                            | IdentKeyWordDecl[@ : "%S(Keyword:literal %S)" "]
                            | AnyTermKeyWordDecl[@ : "%S(Keyword:literal %S)" "]
                            ;

/*  Transformation  Rules  */
transform literal_decl on "show"  <literal_decl> : LiteralDecl(<keyword_decl>,
```

```
                                                        <op_list_of_literal_decl>)
                                                    ;
transform keyword_decl on "Reserved_Word_Decl" <keyword_decl>:
                                ResKeyWordDecl(ResKeyWord("RESERVED_WORD_TERM")),
                          on "Special_Symb_Decl"  <keyword_decl>:
                                SpecSymbKeyWordDecl(SpecSymbKeyWord("SPECIAL_SYMBOL_TERM")),
                          on "Integer_Decl"     <keyword_decl>:
                                IntKeyWordDecl(IntKeyWord("INT_TERM")),
                          on "Real_Decl"        <keyword_decl>:
                                RealKeyWordDecl(RealKeyWord("REAL_TERM")),
                          on "Identifier_Decl" <keyword_decl>:
                                IdentKeyWordDecl(IdentKeyWord("IDENTIFIER_TERM")),
                          on "AnyTerm_Decl"     <keyword_decl>:
                                AnyTermKeyWordDecl(AnyTermKeyWord("ANY_TERM"))
                                                            ;
```

```
/*****************************************************************
*      SSL Specification for  Structured Interpreters            *
*****************************************************************/

/* Abstract  Syntax */
structured_defn          : StructuredDefn(interprtr_name  interprtr_defn)
                         ;

interprtr_name           : InterprtrNameNil()
                         | InterprtrName(ALPHANUMERIC)
                         ;

interprtr_defn           : InterprtrDefnNil()
                         | InterprtrDefn(mandatory_defn
                                         optional_list_of_defns)
                         ;

mandatory_defn           : MandatoryDefn(list_of_rhs_interprtr_names
                                         list_of_semantic_rules)
                         ;

list list_of_rhs_interprtr_names;
list_of_rhs_interprtr_names  : ListofRhsInterprtrNamesNil()
                             | ListofRhsInterprtrNames(rhs_interprtr_name
                                          list_of_rhs_interprtr_names)
                             ;
rhs_interprtr_name           : RhsInterprtrNameNil()
                             | RhsInterprtrName(ALPHANUMERIC)
                             ;
optional list list_of_semantic_rules;
list_of_semantic_rules     : ListofSemanticRulesNil()
                           | ListofSemanticRules(semantic_rule
                                                 list_of_semantic_rules)
                           ;
semantic_rule              : SemanticRuleNil()
                           | InitSemanticRule(init_rule)
                           | CopySemanticRule(copy_rule)
                           | ApplSemanticRule(appl_rule)
                           ;

optional list optional_list_of_defns;
optional_list_of_defns   : OptionalListofDefnsNil()
                         | OptionalListofDefns(op_defn
                                               optional_list_of_defns)
                         ;
op_defn                  : OptionalDefn(separator  mandatory_defn)
                         ;
init_rule                : InitRuleNil()
                         | SynhInitRule(synh_lhs_attr_type  attr_name const)
                         | InhInitRule(inh_lhs_attr_type attr_name const)
                         ;
copy_rule                : CopyRuleNil()
                         | SynhCopyRule(synh_lhs_attr_type  inh_or_synh_type)
                         | InhCopyRule(inh_lhs_attr_type    inh_or_synh_type)
                         ;
appl_rule                : ApplRuleNil()
                         | SynhApplRule(synh_lhs_attr_type  func_name
                                        list_of_rhs_attr_type)
                         | InhApplRule(inh_lhs_attr_type  func_name  list_of_rhs_attr_type)
                         ;
list list_of_rhs_attr_type;
list_of_rhs_attr_type    : ListofRhsAttrTypeNil()
                         | ListofRhsAttrType(inh_or_synh_type
                                             list_of_rhs_attr_type)
                         ;
inh_or_synh_type         : InhOrSynhTypeNil()
                         | InhType(inh_rhs_attr_type)
```

```
                        | SynhType(synh_rhs_attr_type)
                        ;

synh_lhs_attr_type      : SynhLhsAttrType(attr_name)
                        ;

synh_rhs_attr_type      : SynhRhsAttrTypeNil()
                        | SynhRhsAttrTypeRhsIntr(attr_name  interprtr_number)
                        | SynhRhsAttrTypeLhsIntr(attr_name)
                        ;

inh_lhs_attr_type       : InhLhsAttrType(attr_name  interprtr_number)
                        ;

inh_rhs_attr_type       : InhRhsAttrType(attr_name)
                        ;

interprtr_number    : InterprtrNumNil()
                    | InterprtrNum(INT)
                    ;

func_name               : FuncNameNil()
                        | FuncName(ALPHANUMERIC)
                        ;

const                   : ConstNil()
                        | ConstSymb(SYMBOL)
                        | ConstResWord(RESWORD)
                        | ConstCap(ATTRIBUTE)
                        | ConstAlphaNum(ALPHANUMERIC)
                        | ConstInt(INTEGER)
                        | ConstReal(FLOAT)
                        | ConstAnyThing(TERM)
                        ;

/*  Semantic rules  */
structured_defn     : StructuredDefn{interprtr_name.intrprtr_list =
                                structured_defn$1.intrprtr_list;
                      interprtr_defn.intrprtr_list = InsertListInterprtrs(
                         structured_defn$1.name_of_interprtr,
                         structured_defn$1.intrprtr_list);
                      structured_defn$1.name_of_interprtr =
                         interprtr_name.name_of_interprtr;
                      local ALPHANUMERIC lhs_name;
                      lhs_name =  interprtr_name.name_of_interprtr;
                      structured_defn$1.attr_list = interprtr_defn.attr_list;
                         local STR error;
                         error = (ChkLeftRecursion(interprtr_name.name_of_interprtr,
                                  interprtr_defn.lr_interprtr_names))
                               ? "(DEFN. CONTAINS LEFT RECURSION)" : "";
                      interprtr_defn.int_inh = structured_defn$1.int_inh;
                      interprtr_defn.frac_inh = 1;
                      structured_defn$1.int_synh = structured_defn$1.int_inh;
                      /*  INH  */
                      structured_defn$1.u_inh_attr_list = interprtr_defn.u_inh_attr_list;};

interprtr_name      : InterprtrNameNil{interprtr_name$1.name_of_interprtr = "";}
                    | InterprtrName{
                       interprtr_name$1.name_of_interprtr = ALPHANUMERIC;
                       local STR error;
                       error = ChkIntrprtrDef(interprtr_name.name_of_interprtr,
                                              interprtr_name$1.intrprtr_list)
                             ? "(INTERPRETER ALREADY DEFINED)" : "";};

interprtr_defn      : InterprtrDefnNil{interprtr_defn$1.attr_list = AttribListNil;
                                  interprtr_defn$1.lr_interprtr_names =
                                                    LrInterprtrNamesNil;
                                  /*  INH  */
                                  interprtr_defn$1.u_inh_attr_list = AttribListNil;}
                    | InterprtrDefn{optional_list_of_defns.inhopt_attr_list =
                                                    mandatory_defn.mand_attr_list;
                                interprtr_defn$1.attr_list =
                                                    mandatory_defn.mand_attr_list;
                      mandatory_defn.intrprtr_list =
```

```
                                        interprtr defn$1.interprtr list;
                        optional list of defns.interprtr list
                                        interprtr defn$1.interprtr list;
                        interprtr_defn$1.lr interprtr names
                                mandatory_defn.name of interprtr
                                ::
                                optional list of defns.lr interprtr names;
                        mandatory defn.int inh   interprtr defn$1.int inh;
                        optional list of defns.int inh   interprtr defn$1.int inh;
                        mandatory defn.frac inh   interprtr defn$1.frac inh;
                        optional list of defns.frac inh
                                        mandatory defn.frac synh;
                        /*  INH  */
                        interprtr defn$1.u inh attr list
                                optional list of defns.u inh attr list;
                        mandatory defn.d inh attr list   AttribListNil;
                        optional list of defns.d inh attr list
                                mandatory defn.u inh attr list;};

mandatory_defn          : MandatoryDefn(list_of_semantic_rules.inh_attr_list =
                                        AttribListNil;
                                mandatory_defn$1.mand_attr_list
                                    list_of_semantic_rules.synh_attr_list;
                                list_of_rhs_interprtr_names.intrprtr_list =
                                        mandatory_defn$1.intrprtr_list;
                                list_of_semantic_rules.loc_list_intrprtrs =
                                   list_of_rhs_interprtr_names.loc_list_intrprtrs;
                                mandatory_defn$1.name_of_interprtr =
                                  list_of_rhs_interprtr_names.name_of_interprtr;
                                list_of_rhs_interprtr_names.num   1;
                        list_of_semantic_rules.int_inh = mandatory_defn$1.int_inh;
                        mandatory_defn.frac_synh = list_of_semantic_rules.frac_synh;
                        list_of_semantic_rules.frac_inh = mandatory_defn$1.frac_inh;
                        /*  INH */
                        list_of_semantic_rules.d_inh_attr_list =
                            mandatory_defn$1.d_inh_attr_list;
                        mandatory_defn$1.u_inh_attr_list =
                                list_of_semantic_rules.u_inh_attr_list;
                        list_of_rhs_interprtr_names.laip =
                                list_of_semantic_rules.laip;};

list_of_rhs_interprtr_names : ListofRhsInterprtrNamesNil(
                                list_of_rhs_interprtr_names$1.loc_list_intrprtrs =
                                                        LocListInterprtrsNil;
                                list_of_rhs_interprtr_names$1.name_of_interprtr = "";)
                            | ListofRhsInterprtrNames(
                                list_of_rhs_interprtr_names$2.intrprtr_list =
                                    list_of_rhs_interprtr_names$1.intrprtr_list ;
                                local STR error;
                                error = (ChkListInterprtrs
                                        (rhs_interprtr_name.name_of_interprtr,
                                         list_of_rhs_interprtr_names$1.intrprtr_list))
                                      ? "" : "(UNDEFINED)";
                                list_of_rhs_interprtr_names$1.loc_list_intrprtrs =
                                    (rhs_interprtr_name.name_of_interprtr != "")
                                     ?
                                     LocListInterprtrs(Pair(
                                         list_of_rhs_interprtr_names$1.num,
                                         rhs_interprtr_name.name_of_interprtr),
                                         list_of_rhs_interprtr_names$2.loc_list_intrprtrs)
                                     : list_of_rhs_interprtr_names$2.loc_list_intrprtrs;
                                list_of_rhs_interprtr_names$1.name_of_interprtr =
                                            rhs_interprtr_name.name_of_interprtr;
                                list_of_rhs_interprtr_names$2.num =
                                            list_of_rhs_interprtr_names$1.num + 1;
                                /* INH */
                                list_of_rhs_interprtr_names$2.laip =
                                    list_of_rhs_interprtr_names$1.laip;
                                local ATTRIB_LIST attrib_list1;
```

```
                                attrib_list1    ((rhs_interprtr_name.name_of_interprtr != "") &&
                                 (ChkInhAttrList(rhs_interprtr_name.name_of_interprtr,
                                                 {DeclofInterpreters.h}) == true))
                                 ? GetAttribList1(rhs_interprtr_name.name_of_interprtr,
                                                 {DeclofInterpreters.h})
                                 : AttribListNil;
                                local ATTRIB_LIST attrib_list2;
                                attrib_list2 = (rhs_interprtr_name.name_of_interprtr != "")
                                   ? GetAttribList2(list_of_rhs_interprtr_names$1.num,
                                                    list_of_rhs_interprtr_names$1.laip)
                                   : AttribListNil;
                                local STR error1;
                                error1 = ChkAttribLists(attrib_list1, attrib_list2)
                                        ? "{MISSING INH. ATTR. EQN.}"
                                        : "";}; /* all elements of all must be members of al2 */

rhs_interprtr_name : RhsInterprtrNameNil{ rhs_interprtr_name$1.name_of_interprtr = "";}
                   | RhsInterprtrName{ rhs_interprtr_name$1.name_of_interprtr =
                                                              ALPHANUMERIC;};

optional_list_of_defns : OptionalListofDefnsNil{
                                optional_list_of_defns$1.lr_interprtr_names =
                                                              LrInterprtrNamesNil;
                                /*  INH  */
                                optional_list_of_defns$1.u_inh_attr_list =
                                                    optional_list_of_defns$1.d_inh_attr_list;}
                       | OptionalListofDefns{
                           local STR error;
                           error = (ChkMandOptListDefns(op_defn.attr_list,
                                                optional_list_of_defns$1.inhopt_attr_list))
                                  ? "" : "{INCOMP. SYNH. ATTR.}";
                           optional_list_of_defns$2.inhopt_attr_list =
                                  optional_list_of_defns$1.inhopt_attr_list;
                           optional_list_of_defns$2.intrprtr_list =
                                  optional_list_of_defns$1.intrprtr_list;
                           op_defn.intrprtr_list = optional_list_of_defns$1.intrprtr_list;
                           optional_list_of_defns$1.lr_interprtr_names =
                                  op_defn.name_of_interprtr
                                  ::
                                  optional_list_of_defns$2.lr_interprtr_names;
                      op_defn.int_inh = optional_list_of_defns$1.int_inh;
                      op_defn.frac_inh = optional_list_of_defns$1.frac_inh;
                      optional_list_of_defns$2.int_inh =
                                optional_list_of_defns$1.int_inh;
                      optional_list_of_defns$2.frac_inh = op_defn.frac_synh;
                      /*  INH  */
                      op_defn.d_inh_attr_list = optional_list_of_defns$1.d_inh_attr_list;
                      optional_list_of_defns$2.d_inh_attr_list = op_defn.u_inh_attr_list;
                      optional_list_of_defns$1.u_inh_attr_list =
                          optional_list_of_defns$2.u_inh_attr_list;};


op_defn       : OptionalDefn{
                    op_defn$1.attr_list = mandatory_defn.mand_attr_list;
                    mandatory_defn.intrprtr_list = op_defn$1.intrprtr_list;
                    op_defn$1.name_of_interprtr = mandatory_defn.name_of_interprtr;
                    mandatory_defn.int_inh = op_defn$1.int_inh;
                    mandatory_defn.frac_inh = op_defn$1.frac_inh;
                    op_defn$1.frac_synh = mandatory_defn.frac_synh;
                    /*  INH  */
                    op_defn$1.u_inh_attr_list = mandatory_defn.u_inh_attr_list;
                    mandatory_defn.d_inh_attr_list = op_defn$1.d_inh_attr_list;};

list_of_semantic_rules  : ListofSemanticRulesNil{
                            list_of_semantic_rules$1.synh_attr_list =
                                          list_of_semantic_rules$1.inh_attr_list;
                            list_of_semantic_rules$1.frac_synh =
                                            list_of_semantic_rules$1.frac_inh;
                            /*  INH  */
                            list_of_semantic_rules$1.u_inh_attr_list =
```

```
                                        list_of_semantic_rules$1.d_inh_attr_list;
                                        list_of_semantic_rules$1.laip
                                                    listAttrIntPair:Nil;}
                | ListofSemanticRules{
                    list_of_semantic_rules$1.synh_attr_list
                                        list_of_semantic_rules$2.synh_attr_list;
                    list_of_semantic_rules$2.inh_attr_list
                                        InsertAttrList(semantic_rule.attr,
                                        list_of_semantic_rules$1.inh_attr_list);
                    local STR error1;
                    error1 = (ChkAttrList(semantic_rule.attr,
                                        list_of_semantic_rules$1.inh_attr_list))
                          ? "(ATTR MULTIPLY DEFINED)" : " ";
                    semantic_rule.loc_list_intrprtrs =
                            list_of_semantic_rules$1.loc_list_intrprtrs;
                    list_of_semantic_rules$2.loc_list_intrprtrs =
                            list_of_semantic_rules$1.loc_list_intrprtrs;
                    semantic_rule.int_inh = list_of_semantic_rules$1.int_inh;
                    semantic_rule.frac_inh = list_of_semantic_rules$1.frac_inh;
                    list_of_semantic_rules$2.int_inh =
                                list_of_semantic_rules$1.int_inh;
                    list_of_semantic_rules$2.frac_inh
                        list_of_semantic_rules$1.frac_inh + 1;
                    list_of_semantic_rules$1.frac_synh
                        list_of_semantic_rules$2.frac_synh;
                   /*  INH  */
                   list_of_semantic_rules$1.u_inh_attr_list
                        list_of_semantic_rules$2.u_inh_attr_list;
                   semantic_rule.d_inh_attr_list =
                        list_of_semantic_rules$1.d_inh_attr_list;
                   list_of_semantic_rules$2.d_inh_attr_list =
                        semantic_rule.u_inh_attr_list;
                   local STR error2;
                   error2 = ChkAttrIntPair(semantic_rule.aip,
                                        list_of_semantic_rules$2.laip)
                         ? "(ATTR MULTIPLY DEFINED)" : "";
                   list_of_semantic_rules$1.laip =
                        ListAttrIntPair(semantic_rule.aip,
                                    list_of_semantic_rules$2.laip);};

semantic_rule   : SemanticRuleNil{semantic_rule$1.attr = AttribNil;
                                  /* INH */
                                  semantic_rule$1.u_inh_attr_list =
                                    semantic_rule$1.d_inh_attr_list;
                                  semantic_rule$1.aip = AttrIntPairNil;}
                | InitSemanticRule{semantic_rule$1.attr = init_rule.attr;
                                  init_rule.int_inh = semantic_rule$1.int_inh;
                                  init_rule.frac_inh = semantic_rule$1.frac_inh;
                                  init_rule.loc_list_intrprtrs =
                                            semantic_rule$1.loc_list_intrprtrs;
                                  /* INH */
                                  semantic_rule$1.u_inh_attr_list =
                                    semantic_rule$1.d_inh_attr_list;
                                  semantic_rule$1.aip = init_rule.aip;}
                | CopySemanticRule{semantic_rule$1.attr = copy_rule.attr;
                                  copy_rule.loc_list_intrprtrs =
                                            semantic_rule$1.loc_list_intrprtrs;
                                  copy_rule.int_inh = semantic_rule$1.int_inh;
                                  copy_rule.frac_inh = semantic_rule$1.frac_inh;
                                  /*  INH  */
                                  copy_rule.d_inh_attr_list =
                                      semantic_rule$1.d_inh_attr_list;
                                  semantic_rule$1.u_inh_attr_list =
                                      copy_rule.u_inh_attr_list;
                                  semantic_rule$1.aip = copy_rule.aip;}
                | ApplSemanticRule{semantic_rule$1.attr = appl_rule.attr;
                                  appl_rule.loc_list_intrprtrs =
                                            semantic_rule$1.loc_list_intrprtrs;
```

```
                                appl_rule.int_inh   = semantic_rule$1.int_inh;
                                appl_rule.frac_inh  = semantic_rule$1.frac_inh;
                                                    /*  INH  */
                                appl_rule.d_inh_attr_list =
                                    semantic_rule$1.d_inh_attr_list;
                                semantic_rule$1.u_inh_attr_list =
                                    appl_rule.u_inh_attr_list;
                                semantic_rule$1.aip = appl_rule.aip;};

init_rule       : InitRuleNil{init_rule$1.attr = AttribNil;
                              /*  INH  */
                              init_rule$1.aip = AttrIntPairNil;}
                | SynhInitRule{init_rule$1.attr = synh_lhs_attr_type.attr;
                               local STR error1;
                               error1 = ChkAttrDefnTable(attr_name.attr, {Prog.attr_defn_table})
                                        ? "" : "{UNDEFINED}";
                               local STR error2;
                               error2 = ((synh_lhs_attr_type.attr != AttribNil) &&
                                    (attr_name.attr != AttribNil) &&
                                    (synh_lhs_attr_type.attr != attr_name.attr))
                                   ? "{TYPE MISMATCH}" : "";
                               /* INH */
                               init_rule$1.aip = AttrIntPairNil;}
                | InhInitRule{init_rule$1.attr = AttribNil;
                              inh_lhs_attr_type.loc_list_intrprtrs = init_rule$1.loc_list_intrprtrs;
                              local STR error1;
                               error1 = ChkAttrDefnTable(attr_name.attr, {Prog.attr_defn_table})
                                        ? "" : "{UNDEFINED}";
                               local STR error2;
                               error2 = ((inh_lhs_attr_type.inh_lhs_attr != AttribNil) &&
                                    (attr_name.attr != AttribNil) &&
                                    (inh_lhs_attr_type.inh_lhs_attr != attr_name.attr))
                                   ? "{TYPE MISMATCH}" : "";
                              /* INH */
                              init_rule$1.aip = inh_lhs_attr_type.aip;};

copy_rule       : CopyRuleNil{copy_rule$1.attr = AttribNil;
                              /* INH */
                              copy_rule$1.u_inh_attr_list =
                               copy_rule$1.d_inh_attr_list;
                              copy_rule$1.aip = AttrIntPairNil;}
                | SynhCopyRule{copy_rule$1.attr = synh_lhs_attr_type.attr;
                               inh_or_synh_type.loc_list_intrprtrs =
                                                   copy_rule$1.loc_list_intrprtrs;
                        local STR error1;
                        error1 = ((synh_lhs_attr_type.attr != AttribNil) &&
                                  (inh_or_synh_type.attr != AttribNil) &&
                                  (synh_lhs_attr_type.attr != inh_or_synh_type.attr))
                                  ? "{TYPE MISMATCH}" : "";
                        local STR error2;
                        error2 = ((synh_lhs_attr_type.attr != AttribNil) &&
                                   (inh_or_synh_type.inh_rhs_attr != AttribNil) &&
                                   (synh_lhs_attr_type.attr != inh_or_synh_type.inh_rhs_attr))
                                 ? "{TYPE MISMATCH}" : "";
                        local STR error3;
                        error3 = ((synh_lhs_attr_type.attr != AttribNil) &&
                                   (inh_or_synh_type.circ_lhs_attr != AttribNil) &&
                                   (synh_lhs_attr_type.attr == inh_or_synh_type.circ_lhs_attr))
                                 ? "{CIRC. ATTR. DEFN.}" : "";
                        /*  INH  */
                        copy_rule$1.u_inh_attr_list =
                             inh_or_synh_type.u_inh_attr_list;
                        inh_or_synh_type.d_inh_attr_list =
                             copy_rule$1.d_inh_attr_list;
                        copy_rule$1.aip = AttrIntPairNil;}
                | InhCopyRule{copy_rule$1.attr = AttribNil;
                        inh_lhs_attr_type.loc_list_intrprtrs = copy_rule$1.loc_list_intrprtrs;
                        inh_or_synh_type.loc_list_intrprtrs = copy_rule$1.loc_list_intrprtrs;
                        local STR error1;
                        error1 = ((inh_lhs_attr_type.inh_lhs_attr != AttribNil) &&
                                  (inh_or_synh_type.inh_rhs_attr != AttribNil) &&
```

```
                                (inh_lhs_attr_type.inh_lhs_attr != inh_or_synh_type.inh_rhs_attr))
                                ? "{TYPE MISMATCH}" : "";
                        local STR error2;
                        error2  ((inh_lhs_attr_type.inh_lhs_attr != AttribNil) &&
                                (inh_or_synh_type.attr != AttribNil) &&
                                (inh_lhs_attr_type.inh_lhs_attr != inh_or_synh_type.attr))
                                ? "{TYPE MISMATCH}" : "";
                          /*  INH  */
                         copy_rule$1.u_inh_attr_list =
                             inh_or_synh_type.u_inh_attr_list;
                         inh_or_synh_type.d_inh_attr_list =
                             copy_rule$1.d_inh_attr_list;
                         copy_rule$1.aip = inh_lhs_attr_type.aip;};

appl_rule       : ApplRuleNil{appl_rule$1.attr = AttribNil;
                          /*  INH  */
                         appl_rule$1.u_inh_attr_list =
                              appl_rule$1.d_inh_attr_list;
                         appl_rule$1.aip = AttrIntPairNil;}
                | SynhApplRule{appl_rule$1.attr = synh_lhs_attr_type.attr;
                         list_of_rhs_attr_type.loc_list_intrprtrs =
                                              appl_rule$1.loc_list_intrprtrs;
                         local STR error;
                         error = ChkAttrList(synh_lhs_attr_type.attr,
                                        list_of_rhs_attr_type.circ_lhs_attr_list)
                               ? "{CIRC. ATTR. DEFN.}" : "";
                        /*  INH  */
                         list_of_rhs_attr_type.d_inh_attr_list =
                                 appl_rule$1.d_inh_attr_list;
                         appl_rule$1.u_inh_attr_list =
                              list_of_rhs_attr_type.u_inh_attr_list;
                         appl_rule$1.aip = AttrIntPairNil;}
                | InhApplRule{appl_rule$1.attr = AttribNil;
                         list_of_rhs_attr_type.loc_list_intrprtrs =
                                              appl_rule$1.loc_list_intrprtrs;
                         inh_lhs_attr_type.loc_list_intrprtrs = appl_rule$1.loc_list_intrprtrs;
                           /* INH */
                         list_of_rhs_attr_type.d_inh_attr_list =
                                 appl_rule$1.d_inh_attr_list;
                         appl_rule$1.u_inh_attr_list =
                              list_of_rhs_attr_type.u_inh_attr_list;
                         appl_rule$1.aip = inh_lhs_attr_type.aip;};

list_of_rhs_attr_type : ListofRhsAttrTypeNil{ list_of_rhs_attr_type$1.circ_lhs_attr_list =
                                                        AttribListNil;
                                        /* INH */
                                        list_of_rhs_attr_type$1.u_inh_attr_list =
                                         list_of_rhs_attr_type$1.d_inh_attr_list;}
                      | ListofRhsAttrType{inh_or_synh_type.loc_list_intrprtrs =
                                          list_of_rhs_attr_type$1.loc_list_intrprtrs;
                           list_of_rhs_attr_type$2.loc_list_intrprtrs =
                                       list_of_rhs_attr_type$1.loc_list_intrprtrs;
                           list_of_rhs_attr_type$1.circ_lhs_attr_list =
                                        AttribList(inh_or_synh_type.circ_lhs_attr,
                                                list_of_rhs_attr_type$2.circ_lhs_attr_list);
                         /*  INH  */
                         inh_or_synh_type.d_inh_attr_list =
                             list_of_rhs_attr_type$1.d_inh_attr_list;
                         list_of_rhs_attr_type$2.d_inh_attr_list =
                             inh_or_synh_type.u_inh_attr_list;
                         list_of_rhs_attr_type$1.u_inh_attr_list =
                              list_of_rhs_attr_type$2.u_inh_attr_list;};

inh_or_synh_type  : InhOrSynhTypeNil{inh_or_synh_type$1.attr = AttribNil;
                                inh_or_synh_type$1.circ_lhs_attr = AttribNil;
                                /* INH */
                                inh_or_synh_type$1.inh_rhs_attr = AttribNil;
                                inh_or_synh_type$1.u_inh_attr_list =
                                 inh_or_synh_type$1.d_inh_attr_list;}
```

```
                | SynhType{synh_rhs_attr_type.loc_list_intrprtrs =
                                        inh_or_synh_type$1.loc_list_intrprtrs;
                        inh_or_synh_type$1.attr = synh_rhs_attr_type.attr;
                        inh_or_synh_type$1.circ_lhs_attr = synh_rhs_attr_type.circ_lhs_attr;
                        /* INH */
                        inh_or_synh_type$1.inh_rhs_attr = AttribNil;
                                inh_or_synh_type$1.u_inh_attr_list =
                                 inh_or_synh_type$1.d_inh_attr_list;}
                | InhType{inh_or_synh_type$1.attr = AttribNil;
                        inh_or_synh_type$1.circ_lhs_attr = AttribNil;
                        /*  INH  */
                        inh_or_synh_type$1.inh_rhs_attr = inh_rhs_attr_type.inh_rhs_attr;
                        inh_or_synh_type$1.u_inh_attr_list =
                             InsertAttrList(inh_rhs_attr_type.inh_rhs_attr,
                                        inh_or_synh_type$1.d_inh_attr_list);};

synh_lhs_attr_type  : SynhLhsAttrType{synh_lhs_attr_type$1.attr = attr_name.attr;
                                local STR error;
                                error = (ChkAttrDefnTable(attr_name.attr,
                                                {Prog.attr_defn_table}))
                                        ? "" : "{UNDEFINED}";};

synh_rhs_attr_type : SynhRhsAttrTypeNil{synh_rhs_attr_type$1.attr = AttribNil;
                                synh_rhs_attr_type$1.circ_lhs_attr = AttribNil;}
                   | SynhRhsAttrTypeRhsIntr{
                     local STR error1;
                     local STR error2;
                     local STR error3;
                     local ALPHANUMERIC intrprtr_name;
                     intrprtr_name = GetIntrprtrName(interprtr_number.num,
                                synh_rhs_attr_type$1.loc_list_intrprtrs);
                     error1 = ChkSynhAttrList(intrprtr_name, attr_name.attr,
                                                {DeclofInterpreters.a})
                             ? "{ERROR}" : "";
                     error2 = ChkLocListInterprtrs(interprtr_number.num,
                                synh_rhs_attr_type$1.loc_list_intrprtrs)
                             ? "" : "{UNDEFINED INTR.}";
                     error3 = (ChkAttrDefnTable(attr_name.attr,
                                                {Prog.attr_defn_table}))
                             ? "" : "{UNDEFINED}";
                     synh_rhs_attr_type$1.attr = attr_name.attr;
                     synh_rhs_attr_type$1.circ_lhs_attr = AttribNil;}
                   | SynhRhsAttrTypeLhsIntr{
                     local STR error1;
                     local STR error2;
                     error1 = ChkSynhAttrList({StructuredDefn.lhs_name}, attr_name.attr,
                                                {DeclofInterpreters.a})
                             ? "{ERROR}" : "";
                     error2 = (ChkAttrDefnTable(attr_name.attr,
                                                {Prog.attr_defn_table}))
                             ? "" : "{UNDEFINED}";
                     synh_rhs_attr_type$1.attr = attr_name.attr;
                     synh_rhs_attr_type$1.circ_lhs_attr = attr_name.attr; };

inh_lhs_attr_type : InhLhsAttrType{
                     local STR error1;
                     local STR error2;
                     error1 = ChkLocListInterprtrs(interprtr_number.num,
                                inh_lhs_attr_type$1.loc_list_intrprtrs)
                             ? "" : "{UNDEFINED INTR.}";
                     error2 = (ChkAttrDefnTable(attr_name.attr,
                                                {Prog.attr_defn_table}))
                             ? "" : "{UNDEFINED}";
                     /*  INH */
                     inh_lhs_attr_type$1.inh_lhs_attr = attr_name.attr;
                     inh_lhs_attr_type$1.aip = AttrIntPair(attr_name.attr,
                                                        interprtr_number.num);};

inh_rhs_attr_type : InhRhsAttrType{local STR error;
                                error = (ChkAttrDefnTable(attr_name.attr,
                                                {Prog.attr_defn_table}))
```

```
                                        ? "" : "{UNDEFINED}";
                                    /*  INH  */
                                    inh_rhs_attr_type$1.inh_rhs_attr    attr_name.attr;};

interprtr_number : InterprtrNumNil{interprtr_number$1.num    0;}
                 | InterprtrNum{interprtr_number$1.num    INT;};

/*  Function  Definitions  */
BOOL ChkIntrprtrDef(ALPHANUMERIC n, LIST_INTERPRTRS v) {

       with(v) {
         ListInterprtrsNil : false,
         ListInterprtrs(h, t) : (h == n) ? true : ChkIntrprtrDef(n, t)
              }
};

ATTRIB_LIST  InsertAttrList(ATTRIB v, ATTRIB_LIST l) {

          (v != AttribNil)
          ? with(l) {
                AttribListNil : v::l,
                AttribList(hd, tl) : (hd > v) ? v::l
                                   : (hd == v) ? hd::tl
                                   :             hd::InsertAttrList(v, tl)
            }
          : l
   };


BOOL ChkAttrList(ATTRIB v, ATTRIB_LIST l) {

          with(l) {
            AttribListNil : false,
            AttribList(ATTRIB, tl) : ((ATTRIB == v) && (v != AttribNil))
                                   ? true : ChkAttrList(v, tl)
          }
  };

BOOL  ChkMandOptListDefns(ATTRIB_LIST s1, ATTRIB_LIST s2) {

           with(s2) {
             AttribListNil : (s1 == AttribListNil) ? true : false,
             AttribList(h2, t2) : with(s1) {
                AttribListNil  : false,
                AttribList(h1, t1) : (h1 == h2) ? ChkMandOptListDefns(t1, t2) : false
              }
            }
};

SYNH_ATTR_LIST  InsertSynhList(ALPHANUMERIC n, ATTRIB_LIST s1, SYNH_ATTR_LIST s2) {

            ((n != "") && (s1 != AttribListNil))
            ? with(s1) {
               AttribListNil : s2,
               AttribList(ATTRIB, t1) : SynhDecl(n, ATTRIB)::InsertSynhList(n, t1, s2)
                }
             : s2
};

BOOL  ChkListInterprtrs(ALPHANUMERIC n, LIST_INTERPRTRS s1) {

             (n == "") ? true
             :  with(s1) {
                ListInterprtrsNil : false,
                ListInterprtrs(ALPHANUMERIC, t1) : (ALPHANUMERIC == n) ? true
                                                 : ChkListInterprtrs(n, t1)
                }
};

LIST_INTERPRTRS InsertListInterprtrs(ALPHANUMERIC n, LIST_INTERPRTRS s1) {

           (n != "") ? n::s1 : s1
 };

ALPHANUMERIC GetIntrprtrName(INT n1, LOC_LIST_INTERPRTRS s1) {
```

```
            (n1    0) : ""
            : with(s1) {
               LocListInterprtrsNil : "",
               LocListInterprtrs(Pair(n2, a), t1) : (n1    n2) ? a : GetInterprtrName(n1, t1)
            }
};

BOOL  ChkSynhAttrList(ALPHANUMERIC i, ATTRIB a, SYNH_ATTR_LIST s1) {

            ((i !  "") && (a !  AttribNil))
            ? with(s1) {
               SynhAttrListNil : true,
               SynhAttrDecl(SynhDecl(ALPHANUMERIC, ATTRIB), t1) :
                 ((i =  ALPHANUMERIC) && (a    ATTRIB)) ? false : ChkSynhAttrList(i, a, t1)
            }
            : false
};

BOOL  ChkLocListInterprtrs(INT n1, LOC_LIST_INTERPRTRS s1) {

          (n1 == 0) ? true
          : with(s1) {
             LocListInterprtrsNil : false,
             LocListInterprtrs(Pair(n2, ALPHANUMERIC), t1) : (n1    n2) ? true
                                                           : ChkLocListInterprtrs(n1, t1)
          }
};

BOOL ChkLeftRecursion(ALPHANUMERIC n1, LR_INTERPRTR_NAMES s1) {

         ((n1 == "") || (s1 ==  LrInterprtrNamesNil))
         ? false
         : with(s1) {
            LrInterprtrNamesNil : false,
            LrInterprtrNames(ALPHANUMERIC, t1) : (n1    ALPHANUMERIC) ? true
                                               : ChkLeftRecursion(n1, t1)
         }
};

/*  INH  */

BOOL ChkInhAttrList(ALPHANUMERIC n, INH_ATTR_LIST s1) {

    (n != "")
    ? with(s1) {
       InhAttrListNil : false,
       InhAttrList(InhAttr(ALPHANUMERIC, p), t1) : (ALPHANUMERIC  n) ? true
                                                 : ChkInhAttrList(n, t1)
    } : false
};

INH_ATTR_LIST InsertSymbTable(ALPHANUMERIC n, ATTRIB_LIST s1, INH_ATTR_LIST s2) {

      ((n != "") && (s1 != AttribListNil))
      ? InhAttr(n, s1)::s2
      : s2
};

ATTRIB_LIST GetAttribList1(ALPHANUMERIC n, INH_ATTR_LIST s1) {

        with(s1) {
         InhAttrListNil : AttribListNil,
         InhAttrList(InhAttr(ALPHANUMERIC, s), t1) : (ALPHANUMERIC  n) ? s
                                                   : GetAttribList1(n, t1)
        }
};

ATTRIB_LIST  GetAttribList2(INT n, LIST_ATTR_INT_PAIR s1) {

        with(s1) {
          ListAttrIntPairNil : AttribListNil,
          ListAttrIntPair(AttrIntPairNil, t1) : AttribListNil,
          ListAttrIntPair(AttrIntPair(ATTRIB, INT), t1) : (INT == n)
                                                        ? ATTRIB::GetAttribList2(n, t1)
                                                        : GetAttribList2(n, t1)
        }
};

BOOL  ChkAttribLists(ATTRIB_LIST s1, ATTRIB_LIST s2) {
```

```
          ((s1 != AttribListNil) && (s2 == AttribListNil))
          ? true
          : ((s1 == AttribListNil) && (s2 != AttribListNil))
          ? false
          : ((s1 == AttribListNil) && (s2 == AttribListNil))
          ? false
          : with(s1) {
             AttribListNil : false,
             AttribList(ATTRIB, t1) : ChkAttrList(ATTRIB, s2) ? ChkAttribLists(t1, s2)
                                      : true
            }
};

BOOL ChkAttrIntPair(ATTR_INT_PAIR a1, LIST_ATTR_INT_PAIR s1) {

     with(s1) {
      ListAttrIntPairNil : false,
      ListAttrIntPair(a2, s2) : ((a1 != AttrIntPairNil) &&
                                 (a2 != AttrIntPairNil) &&
                                 (a1 == a2)) ? true
                                 : ChkAttrIntPair(a1, s2)
    }
};

/*  Entry  Declarations */

Interprtr_name  { synthesized  interprtr_name  val; };
interprtr_name ^ Interprtr_name.val;
Rhs_interprtr_name { synthesized  rhs_interprtr_name  val; };
rhs_interprtr_name ^ Rhs_interprtr_name.val;
Interprtr_number { synthesized interprtr_number val; };
interprtr_number ^ Interprtr_number.val;
Func_name { synthesized func_name val; };
func_name ^ Func_name.val;
Const { synthesized const val; };
const ^ Const.val;

/*   Parsing  Declarations  */

Interprtr_name ::= (ALPHANUMERIC) {Interprtr_name.val =
                                   InterprtrName(ALPHANUMERIC);}
                ;

Rhs_interprtr_name ::= (ALPHANUMERIC) {Rhs_interprtr_name.val =
                                       RhsInterprtrName(ALPHANUMERIC);}
                    ;

Attr_name ::= (ATTRIBUTE) {Attr_name.val = AttrName(AttribName(ATTRIBUTE));}
           ;

Interprtr_number ::= (INTEGER) {Interprtr_number.val =
                                InterprtrNum(STRtoINT(INTEGER));}
                  ;

Func_name ::= (ALPHANUMERIC) {Func_name.val = FuncName(ALPHANUMERIC);}
           ;

Const  ::=  (SYMBOL)       {Const.val = ConstSymb(SYMBOL);}
         |  (RESWORD)      {Const.val = ConstResWord(RESWORD);}
         |  (ATTRIBUTE)    {Const.val = ConstCap(ATTRIBUTE);}
         |  (ALPHANUMERIC) {Const.val = ConstAlphaNum(ALPHANUMERIC);}
         |  (INTEGER)      {Const.val = ConstInt(INTEGER);}
         |  (FLOAT)        {Const.val = ConstReal(FLOAT);}
         |  (TERM)         {Const.val = ConstAnyThing(TERM);};

/*    Unparsing  Rules  */

structured_defn          : StructuredDefn[^ : ^ error "%n=%t%n" ^]
                         ;

interprtr_name           : InterprtrNameNil[@ ::= "%S(Placeholder:<interprtr_name>%S)"]
                         | InterprtrName[@ ::= ^ error]
                         ;

interprtr_defn           : InterprtrDefnNil[@ ::= "%S(Placeholder:<interprtr_defn>%S)"]
                         | InterprtrDefn[@ : ^ "%n" ^]
                         ;

mandatory_defn           : MandatoryDefn[@ : "%S(Keyword:structure%S) ( " ^ " )%n" "[" ^ "]"]
                         ;

list_of_rhs_interprtr_names : ListofRhsInterprtrNamesNil[@ ::=]
```

```
                        | ListofRhsInterprtrNames
                          [@ :: "s" list of rhs interprtr names%S.nnn " "^
                                error error1 "," &rr+ " "]@
                        ;

rhs_interprtr_name      : RhsInterprtrNameNil[@ ::  "%S(Placeholder:<interprtr name>%S)"]
                        | RhsInterprtrName[@ ::  ^]
                        ;

list_of_semantic_rules  : ListofSemanticRulesNil[@ :: ]
                        | ListofSemanticRules[@ : ^ error1 error2 ";",%n"] @]
                        ;

semantic_rule           : SemanticRuleNil[@ ::  "%S(Placeholder:<semantic rule>%S)"]
                        | InitSemanticRule[@ : ^]
                        | CopySemanticRule[@ : ^]
                        | ApplSemanticRule[@ : ^]
                        ;

init_rule               : InitRuleNil[@ ::  "%S(Placeholder:<init rule>%S)"]
                        | SynhInitRule[^ : "i rule " init rule.int inh","init rule.frac inh
                                " (" ^ ")" " EQ " "(" error2 ^ error1 " " ^ ")"]
                        | InhInitRule[^ : "i rule " init rule.int inh","init rule.frac inh
                                " (" ^ ")" " EQ " "(" error2 ^ error1 " " ^ ")"]
                        ;

copy_rule               : CopyRuleNil[@ :: "%S(Placeholder:<copy rule>%S)"]
                        | SynhCopyRule[^ : "c rule " copy rule.int inh","copy rule.frac inh
                                " (" ^ ")" " EQ " "(" ^ ")" error1 error2 error3]
                        | InhCopyRule[^ : "c rule " copy rule.int inh","copy rule.frac inh
                                " (" ^ ")" " EQ " "(" ^ ")" error1 error2]
                        ;

appl_rule               : ApplRuleNil[@ :: "%S(Placeholder:<appl rule>%S)"]
                        | SynhApplRule[^ : "a rule " appl rule.int inh","appl rule.frac inh
                                " (" ^ ")" " EQ " ^ "[" ^ "]" error1]
                        | InhApplRule[^ : "a rule " appl rule.int inh","appl rule.frac inh
                                " (" ^ ")" " EQ " ^ "[" ^ "]"]
                        ;

list_of_rhs_attr_type   : ListofRhsAttrTypeNil[@ ::]
                        | ListofRhsAttrType[@ : ^ [","] @]
                        ;

inh_or_synh_type        : InhOrSynhTypeNil[@ ::= "%S(Placeholder:<attr type>%S)"]
                        | InhType[^ : ^]
                        | SynhType[^ : ^]
                        ;

synh_lhs_attr_type      : SynhLhsAttrType[@ : error ^ " $u lhs"]
                        ;

synh_rhs_attr_type      : SynhRhsAttrTypeNil[@ ::= "<synthesized attr>"]
                        | SynhRhsAttrTypeRhsIntr[@ : error3 ^ error1 " $u  s" ^ error2]
                        | SynhRhsAttrTypeLhsIntr[@ : error2 ^ error1 " $u lhs"]
                        ;

inh_lhs_attr_type       : InhLhsAttrType[@ : error2 ^  " $d s" ^ error1]
                        ;

inh_rhs_attr_type       : InhRhsAttrType[@ : error ^ " $d lhs"]
                        ;

interprtr_number        : InterprtrNumNil[@ ::= "%S(Placeholder:<int num>%S)"]
                        | InterprtrNum[@ ::= ^]
                        ;

func_name               : FuncNameNil[@ ::= "%S(Placeholder:<func name>%S)"]
                        | FuncName[@ ::= ^]
                        ;

const                   : ConstNil[@ ::= "%S(Placeholder:<const>%S)"]
                        | ConstSymb[@ ::= ^]
                        | ConstResWord[@ ::= ^]
                        | ConstCap[@ ::= ^]
                        | ConstAlphaNum[@ ::= ^]
                        | ConstInt[@ ::= ^]
                        | ConstReal[@ ::= ^]
                        | ConstAnyThing[@ ::= ^]
                        ;

optional_list_of_defns  : OptionalListofDefnsNil[@ ::=]
                        | OptionalListofDefns[@ ::= ^ error ["%n"] @]
                        ;
```

```
op defn                         : OptionalDefn[0 : * "in" *]
                                ;

/* Transformation  Rules  */
transform interprtr_defn on "show" <interprtr_defn> :
                                InterprtrDefn(<mandatory_defn>,
                                              <optional_list_of_defns>)
                                              ;

transform semantic_rule on "i_rule" <semantic_rule> :
                                InitSemanticRule(<init_rule>),

                        on "c_rule" <semantic_rule> :
                                CopySemanticRule(<copy_rule>),
                        on "a_rule" <semantic_rule> :
                                ApplSemanticRule(<appl_rule>)
                                                ;

transform init_rule on "synthesized_rule" <init_rule> :
                                SynhInitRule(<synh_lhs_attr_type>,
                                             <attr_name>,
                                             <const>),

                    on "inherited_rule"   <init_rule> :
                                InhInitRule(<inh_lhs_attr_type>,
                                            <attr_name>, <const>)
                                                ;

transform copy_rule on "synthesized_rule" <copy_rule> :
                                SynhCopyRule(<synh_lhs_attr_type>,
                                             <inh_or_synh_type>),

                    on "inherited_rule"   <copy_rule> :
                                InhCopyRule(<inh_lhs_attr_type>,
                                            <inh_or_synh_type>)
                                                ;

transform appl_rule on "synthesized_rule" <appl_rule> :
                                SynhApplRule(<synh_lhs_attr_type>,
                                             <func_name>, <list_of_rhs_attr_type>),

                    on "inherited_rule"   <appl_rule> :
                                InhApplRule(<inh_lhs_attr_type>,
                                            <func_name>, <list_of_rhs_attr_type>)
                                                ;

transform inh_or_synh_type on "inherited_rule" <inh_or_synh_type> :
                                InhType(<inh_rhs_attr_type>),

                           on "synthesized_rule"   <inh_or_synh_type> :
                                SynhType(<synh_rhs_attr_type>)
                                                ;

transform synh_rhs_attr_type on "rhs_interpreter" <synh_rhs_attr_type> :
                                SynhRhsAttrTypeRhsIntr(<attr_name>, <interprtr_number>),

                             on "lhs_interpreter" <synh_rhs_attr_type> :
                                SynhRhsAttrTypeLhsIntr(<attr_name>)
                                                ;

transform  mandatory_defn on  "sem_rules"  MandatoryDefn(s,ListofSemanticRulesNil) :
       MandatoryDefn(s,ListofSemanticRules(<semantic_rule>,[list_of_semantic_rules]));
```

APPENDIX 6
VITA AUCTORIS

Wadia Farook A. was born in 1969 in Bombay, India. He graduated from Chauhan Institute of Science, Bombay, India in 1986. From there he went on to the University of Poona, Pune, India where he obtained a B. E. in Computer Technology in 1990. He is currently a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in the Spring of 1993.